



A University of Sussex DPhil thesis

Available online via Sussex Research Online:

<http://sro.sussex.ac.uk/>

This thesis is protected by copyright which belongs to the author.

This thesis cannot be reproduced or quoted extensively from without first obtaining permission in writing from the Author

The content must not be changed in any way or sold commercially in any format or medium without the formal permission of the Author

When referring to this work, full bibliographic details including the author, title, awarding institution and date of the thesis must be given

Please visit Sussex Research Online for more information and further details

EVALUATION OF THE USABILITY OF CONSTRAINT DIAGRAMS AS A VISUAL MODELLING LANGUAGE: THEORETICAL AND EMPIRICAL INVESTIGATIONS

NOORA FETAIS

A thesis submitted in partial fulfilment of the requirements of the University of Sussex
for the degree of Doctor of Philosophy

The University of Sussex

September, 2012

Declaration

I hereby declare that this thesis has not been and will not be, submitted in whole or in part to another University for the award of any other degree. After registration for the degree, and under supervision, these are the published paper names which are related to the thesis chapters:

Fetais, Noora and Cheng, Peter C.-H. *Empirical Evaluation of Constraint Diagrams Notation*. Qatar Foundation Annual Research Forum Proceedings (2012), Qatar.

Fetais, Noora and Cheng, Peter C.-H. *Using Cognitive Dimensions Framework to Evaluate Constraint Diagrams*. Qatar Foundation Annual Research Forum Proceedings (2011), Qatar.

Fetais, Noora. *Constraint diagrams can be used to interpret program specification expressions: an evaluation experiment with novice users*. Proceedings of Qatar Foundation Annual Research Forum, Doha (2010), Qatar.

Fetais, Noora and Cheng, Peter C.-H. *An Experiment to Evaluate Constraint Diagrams with Novice Users*. A.K. Goel, M. Jamnik, and N.H. Narayanan (Eds.): Diagrams 2010, LNAI 6170, pp. 307–309, 2010. © Springer-Verlag Berlin Heidelberg 2010.

Signature:

A handwritten signature in brown ink on a yellow background. The signature is stylized, with a large loop at the top and several horizontal strokes below it.

Acknowledgements

First and foremost, profound thanks and gratitude to my supervisor, Professor Peter Cheng at the University of Sussex for being a constant source of advice, encouragement and enthusiasm, and for affording me ‘beyond the call of duty’ amounts of time and endless support. Being Peter’s student has been a great privilege. Without his generous support, this thesis would not have been brought to its conclusion and I would not have had the not inconsiderable courage to expose the fields of cognitive science and representational design. I am thankful for his incredible way of thinking to break a complex task into a number of simple and easy to understand and solve chunks.

Secondly I would like to express my special thanks to Professor John Howse at the University of Brighton for being a constant source of support, advice and expertise in the field of representational notations especially in Constraint Diagrams Language. The continuing influence of John has been appreciated. I also acknowledge the invaluable assistance that Professor Thomas Green gave me with Cognitive Dimensions.

Thirdly, I must also express gratitude to both Dr. Gem Stapleton and Dr. Ian Mackie for agreeing to formally examine the thesis.

Thanks, also, to my annual review examiners, Dr. Richard Cox, Dr. Rudi Lutz, Dr. Bill Keller, and Dr. Ian Mackie for their time, support, advice and encouragement.

I would like to thank all the members of the Cognitive and Language Processing Systems Group (CALPS) at the University of Sussex for the supportive and enthusiastic environment they provided. It has been a privilege to work within this group. Thanks must also go to the individuals who participated in the evaluations for their contributions and patience in spending hours of their time understanding, interpreting and constructing the desired notations.

Finally I would like to express my sincere appreciation to the most important people in my life. To my family, thank you for the support given to me at all times and the encouragement to pursue my dreams.

Evaluation of the Usability of Constraint Diagrams as a Visual Modelling Language: Theoretical and Empirical Investigations.

Noora Fetais

SUMMARY

This research evaluates the constraint diagrams (CD) notation, which is a formal representation for program specification that has some promise to be used by people who are not expert in software design. Multiple methods were adopted in order to provide triangulated evidence of the potential benefits of constraint diagrams compared with other notational systems. Three main approaches were adopted for this research.

The first approach was a semantic and task analysis of the CD notation. This was conducted by the application of the Cognitive Dimensions framework, which was used to examine the relative strengths and weaknesses of constraint diagrams and conventional notations in terms of the perceptive facilitation or impediments of these different representations. From this systematic analysis, we found that CD cognitively reduced the cost of exploratory design, modification, incrementation, searching, and transcription activities with regard to the cognitive dimensions: consistency, visibility, abstraction, closeness of mapping, secondary notation, premature commitment, role-expressiveness, progressive evaluation, diffuseness, provisionality, hidden dependency, viscosity, hard mental operations, and error-proneness.

The second approach was an empirical evaluation of the comprehension of CD compared to natural language (NL) with computer science students. This experiment took the form of a web-based competition in which 33 participants were given instructions and training on either CD or the equivalent NL specification expressions, and then after each example, they responded to three multiple-choice questions requiring the interpretation of expressions in their particular notation. Although the CD group spent more time on the training and had less confidence, they obtained comparable interpretation scores to the NL group and took less time to answer the questions, although they had no prior experience of CD notation.

The third approach was an experiment on the construction of CD. 20 participants were given instructions and training on either CD or the equivalent NL specification expressions, and then after each example, they responded to three questions requiring the construction of expressions in their particular notation. We built an editor to allow the construction of the two notations, which automatically logged their interactions. In general, for constructing program specification, the CD group had more accurate answers, they had spent less time in training, and their returns to the training examples were fewer than those of the NL group.

Overall it was found that CD is understandable, usable, intuitive, and expressive with unambiguous semantic notation.

Submitted for the degree of Doctor of Philosophy

The University of Sussex

September, 2012

Table of Contents

DECLARATION.....	2
ACKNOWLEDGEMENTS.....	3
TABLE OF CONTENTS.....	6
LIST OF TABLES	9
LIST OF FIGURES	10
CHAPTER 1 INTRODUCTION	14
1.1. MOTIVATION AND BACKGROUND.....	14
1.2. RESEARCH GOALS AND OBJECTIVES	17
1.3. RESEARCH METHODOLOGY	18
1.4. OVERVIEW OF THE THESIS	19
1.4.1. <i>Chapter 2: A Review of the Literature</i>	19
1.4.2. <i>Chapter 3: Constraint Diagrams</i>	19
1.4.3. <i>Chapter 4: Cognitive Dimensions Analysis</i>	20
1.4.4. <i>Chapter 5: Experiment 1: Interpretation of Constraint Diagrams</i>	20
1.4.5. <i>Chapter 6: Experiment 2: Construction of Constraint Diagrams</i>	20
1.4.6. <i>Chapter 7: General Conclusions</i>	21
CHAPTER 2 A REVIEW OF THE LITERATURE	22
2.1. INTRODUCTION.....	22
2.2. PROGRAM SPECIFICATION LANGUAGES.....	23
2.3. LOGIC AND PROGRAM SPECIFICATION DIAGRAMS.....	24
2.4. APPROACHES TO SUPPORTING THE LEARNING OF LOGIC.....	26
2.5. THEORY OF DIAGRAMMATIC REPRESENTATIONS	29
2.6. THE CONSTRAINT DIAGRAM FAMILY	30
2.7. METHODOLOGIES AND FRAMEWORKS FOR ANALYSING NOTATIONAL SYSTEMS	33
CHAPTER 3 CONSTRAINT DIAGRAMS	35
3.1. INTRODUCTION.....	35
3.2. SYNTAX.....	36
3.2.1. <i>Informal syntax</i>	36
3.2.2. <i>Formal Syntax</i>	39
3.3. SEMANTICS.....	41
3.3.1. <i>Informal Semantics</i>	41
3.3.2. <i>Formal Semantics</i>	42
3.4. REASONING	42
3.5. FRAGMENTS OF CONSTRAINT DIAGRAMS.....	44

3.6.	USING CONSTRAINT DIAGRAMS IN PROGRAM SPECIFICATION	44
3.6.1.	<i>Patient Population: class PP[P; I]</i>	45
3.6.2.	<i>Health-Care System: class HCS[P; I; S; H; Q]</i>	48
3.6.3.	<i>Patient Record System: class PRS[P; I; S; H; Q; R; F]</i>	53
3.7.	USAGE.....	56
3.8.	COMPARISON WITH OTHER DIAGRAMS.....	57
3.8.1.	<i>Commutative diagram</i>	58
3.8.2.	<i>Higraph</i>	59
3.8.3.	<i>UML diagrams</i>	60
3.9.	DISCUSSION.....	61
CHAPTER 4	COGNITIVE DIMENSIONS OF NOTATIONS FRAMEWORK ANALYSIS ..	63
4.1.	INTRODUCTION.....	63
4.2.	NOTATIONAL ACTIVITIES	65
4.3.	DIMENSIONS	67
4.4.	EXAMPLES FROM THE PATIENT RECORD SYSTEM.....	71
4.5.	PROFILES	78
4.5.1.	<i>Exploratory Design Activity</i>	78
4.5.2.	<i>Modification Activity</i>	86
4.5.3.	<i>Incrementation Activity</i>	91
4.5.4.	<i>Searching Activity</i>	96
4.5.5.	<i>Transcription Activity</i>	100
4.6.	DISCUSSION.....	107
CHAPTER 5	EXPERIMENT 1: INTERPRETATION OF CONSTRAINT DIAGRAMS.....	109
5.1.	INTRODUCTION.....	109
5.2.	EXPERIMENTAL DESIGN	110
5.3.	PILOT EXPERIMENTS	111
5.4.	EXPERIMENT	114
5.4.1.	<i>Method</i>	114
5.4.2.	<i>Subjects</i>	116
5.4.3.	<i>Materials</i>	116
5.5.	RESULTS.....	119
5.5.1.	<i>Correct Answers</i>	121
5.5.2.	<i>Time Spent on each Question</i>	124
5.5.3.	<i>Confidence Rating</i>	126
5.5.4.	<i>Time Spent on each Example</i>	130
5.5.5.	<i>Returns to Example</i>	132
5.5.6.	<i>Relations between different measures</i>	132
5.6.	DISCUSSION.....	133
CHAPTER 6	EXPERIMENT 2: CONSTRUCTION OF CONSTRAINT DIAGRAMS	138

6.1.	INTRODUCTION.....	138
6.2.	EXPERIMENTAL DESIGN	139
6.3.	PILOT EXPERIMENTS	146
6.4.	EXPERIMENT	146
6.4.1.	<i>Method</i>	146
6.4.2.	<i>Subjects</i>	148
6.4.3.	<i>Materials</i>	149
6.5.	RESULTS.....	156
6.5.1.	<i>Correct Answers</i>	157
6.5.2.	<i>Percentage of Steps</i>	168
6.5.3.	<i>Percentage of the Number of Returns to the Examples</i>	170
6.5.4.	<i>Time Spent on each Example</i>	172
6.5.5.	<i>Time Spent on each Initial-Thinking</i>	177
6.5.6.	<i>Relations between different measures</i>	179
6.6.	DISCUSSION.....	181
CHAPTER 7	GENERAL CONCLUSIONS	185
7.1.	INTRODUCTION.....	185
7.2.	IMPLICATIONS OF THE KEY FINDINGS.....	188
7.2.1.	<i>Is CD notation effective for supporting novice users?</i>	189
7.2.2.	<i>What are the relative strengths and weaknesses of CD notation and conventional NL notation?</i>	189
7.2.3.	<i>Is CD a good notation for the interpretation of program specification expressions?....</i>	190
7.2.4.	<i>Is CD a good notation for the construction of program specification expressions?.....</i>	191
7.2.5.	<i>Is CD notation effective for program specification?</i>	191
7.3.	THESIS LIMITATIONS AND FUTURE WORK.....	192
BIBLIOGRAPHY	197
APPENDIX A: MATERIAL USED FOR CONDUCTING EXPERIMENT 1.....		207
APPENDIX B: MATERIAL USED FOR CONDUCTING EXPERIMENT 2.....		221

List of Tables

<i>Table 3.1 These Formal Semantics are based on (Fish, et al., 2005b)</i>	<i>42</i>
<i>Table 4.1 Cognitive Dimensions Definitions.....</i>	<i>68</i>
<i>Table 4.2 Summary of Exploratory Design Activity Profile</i>	<i>86</i>
<i>Table 4.3 Summary of Modification Activity Profile.....</i>	<i>91</i>
<i>Table 4.4 Summary of Incrementation Activity Profile</i>	<i>96</i>
<i>Table 4.5 Summary of Searching Activity Profile</i>	<i>100</i>
<i>Table 4.6 Summary of Transcription Activity Profile.....</i>	<i>104</i>
<i>Table 4.7 CD Integrated Profile</i>	<i>105</i>
<i>Table 4.8 NL Integrated Profile</i>	<i>106</i>
<i>Table 5.1 The Pearson Product Moment Correlation between different measures for CD group.....</i>	<i>133</i>
<i>Table 5.2 The Pearson Product Moment Correlation between different measures for NL group.....</i>	<i>133</i>
<i>Table 5.3 Summary of the Results</i>	<i>135</i>
<i>Table 6.1 The Pearson Product Moment Correlation between different measures for the CD group</i>	<i>180</i>
<i>Table 6.2 The Pearson Product Moment Correlation between different measures for the NL group.....</i>	<i>181</i>
<i>Table 6.3 Summary of the Results where the CD group performed better than the NL group.....</i>	<i>183</i>
<i>Table 6.4 Summary of the Results where the NL group performed as well as or better than the CD group</i>	<i>184</i>
<i>Table A.1 The Examples for the CD group</i>	<i>207</i>
<i>Table A.2 The Examples for the NL group</i>	<i>212</i>
<i>Table A.3 The Questions for both the CD and the NL groups and the correct answer.....</i>	<i>214</i>
<i>Table B.1 The Examples for the CD group</i>	<i>221</i>
<i>Table B.2 The Examples for the NL group</i>	<i>226</i>
<i>Table B.3 The Part A - Questions for both the CD and the NL groups.....</i>	<i>231</i>
<i>Table B.4 The Part B - Questions for both the CD and the NL groups.....</i>	<i>236</i>
<i>Table B.5 The correct answer</i>	<i>241</i>

List of Figures

<i>Figure 1.1 Constraint Diagrams representation of the invariant of the Patient Record System</i>	<i>16</i>
<i>Figure 2.1 OCL representation of the invariant of the Patient Record System.....</i>	<i>23</i>
<i>Figure 2.2 Entity-Relationship Diagram.....</i>	<i>25</i>
<i>Figure 2.3 Harel's Statechart</i>	<i>25</i>
<i>Figure 2.4 diagram to present the existence of an element in patient.</i>	<i>30</i>
<i>Figure 2.5 Spider Diagrams (3 versions).....</i>	<i>32</i>
<i>Figure 2.6 Concepts Patient, PRec, and Communication are represented by the Concept Diagrams</i>	<i>32</i>
<i>Figure 3.1 An example of an ambiguous constraint diagram</i>	<i>37</i>
<i>Figure 3.2 This Formal Syntax is based on (Fish, et al., 2005b).....</i>	<i>39</i>
<i>Figure 3.3 Deleting an existential spider.....</i>	<i>43</i>
<i>Figure 3.4 Class-structure for Patient Record System.....</i>	<i>44</i>
<i>Figure 3.5 Specification of an invariant that uses constraint diagrams</i>	<i>45</i>
<i>Figure 3.6 Constraint Diagrams as an Event to Register Patients</i>	<i>46</i>
<i>Figure 3.7 An event to update Patient Information at the PP level</i>	<i>47</i>
<i>Figure 3.8 An event to record Patient's Death at the PP level.....</i>	<i>47</i>
<i>Figure 3.9 The HCS invariant.....</i>	<i>48</i>
<i>Figure 3.10 An event to define a new service.....</i>	<i>49</i>
<i>Figure 3.11 An event to define a new sub-service.....</i>	<i>49</i>
<i>Figure 3.12 An event to register a new health professional.....</i>	<i>50</i>
<i>Figure 3.13 An event to record a new qualification.....</i>	<i>50</i>
<i>Figure 3.14 An event to associate a health-professional with a (sub-) service</i>	<i>51</i>
<i>Figure 3.15 An event to dissociate a halth-professional from a (sub-) service.....</i>	<i>51</i>
<i>Figure 3.16 An event to enroll a patient into (sub-) service.....</i>	<i>52</i>
<i>Figure 3.17 An event to discharge a patient from (sub-) service</i>	<i>52</i>
<i>Figure 3.18 Two Events that are promoted from the extended invariant.....</i>	<i>52</i>
<i>Figure 3.19 The PRS invariant</i>	<i>53</i>
<i>Figure 3.20 An event to create a new note.....</i>	<i>54</i>
<i>Figure 3.21 An event to send a communication</i>	<i>55</i>
<i>Figure 3.22 Events that are promoted from this or extended invariants.....</i>	<i>56</i>
<i>Figure 3.23 Commutative Diagrams.....</i>	<i>58</i>
<i>Figure 3.24 Higraph</i>	<i>59</i>
<i>Figure 3.25 Two non-intersected blobs.....</i>	<i>60</i>
<i>Figure 3.26 Class diagram.....</i>	<i>60</i>
<i>Figure 3.27 State diagram for HProf.....</i>	<i>61</i>
<i>Figure 4.1 CD diagram to represent the Patient set with two disjoint subsets</i>	<i>71</i>
<i>Figure 4.2 NL statement to represent the Patient set with two disjoint subsets</i>	<i>72</i>

Figure 4.3 CD diagram to represent the Gender set with two disjoint subsets.....	72
Figure 4.4 NL statement to represent that the Gender set with two disjoint subsets	72
Figure 4.5 CD diagram to represent the Patient set with five subsets	73
Figure 4.6 NL statement to represent the Patient set with five subsets	73
Figure 4.7 CD diagram to present the five subsets of the Patient set along with a spider.....	74
Figure 4.8 NL statement to present the five subsets of the Patient set along with a spider	74
Figure 4.9 CD diagram to present the two sets Patient and Gender along with their subset	75
Figure 4.10 NL statement to present the two sets Patient and Gender along with their subsets	75
Figure 4.11 CD diagram to present that each Patient is either a Male or Female only.....	75
Figure 4.12 NL statement to represent that each Patient is either a Male or Female only	76
Figure 4.13 CD diagram to present that each Patient has one or many PRec files.....	76
Figure 4.14 NL statement to represent that each Patient has one or many PRec files	76
Figure 4.15 A CD diagram to represent that each PRec file refers to only one patient	76
Figure 4.16 NL statement to represent that each PRec file refers to only one patient.....	77
Figure 4.17 CD diagram to present the problem of the differences between domain and habitat.....	77
Figure 4.18 NL statement to present the problem of the differences between domain and habitat.....	77
Figure 4.19 A CD diagram to present the generalized version of the CD	77
Figure 4.20 NL statement to represent two relations at the same time	77
Figure 5.1 A snapshot from Pilot 1	112
Figure 5.2 A snapshot of training example 7 using CD	117
Figure 5.3 A snapshot of training example 7 using NL.....	118
Figure 5.4 A snapshot of question 23 using CD.....	118
Figure 5.5 A snapshot of question 23 using NL	119
Figure 5.6 The drop-out rate.....	120
Figure 5.7 Graph of the average of correct answers for the two groups across the 24 questions	122
Figure 5.8 Graph of the interaction of CD and NL representations and correct answers for the two halves of the 24 questions.....	123
Figure 5.9 Graph of the average time for the two groups across the 24 questions.....	125
Figure 5.10 Graph of the interaction of CD and NL representations and the time spent on questions for the two halves of the 24 questions.....	125
Figure 5.11 Graph of the confidence average for the two groups across the 24 questions	126
Figure 5.12 Graph of the interaction of CD and NL representations and the level of the confidence rating for the two halves of the 24 questions	128
Figure 5.13 Graph of the interaction of CD and NL representations and normalized confidence rate the two halves of the 24 questions.....	129
Figure 5.14 Graph of the example time average for the two groups across the 24 questions	131
Figure 5.15 Graph of the interaction of CD and NL representations and the time spent on examples for the two halves of the 24 questions.....	132

<i>Figure 6.1 Evaluating Methods of Program Specification: Constructing CD – Training Question: Part A</i>	<i>141</i>
<i>Figure 6.2 Evaluating Methods of Program Specification: Constructing CD – Training Question: Part B</i>	<i>142</i>
<i>Figure 6.3 Evaluating Methods of Program Specification: Constructing FSL – Training Question: Part A</i>	<i>143</i>
<i>Figure 6.4 Evaluating Methods of Program Specification: Constructing FSL – Training Question: Part B</i>	<i>144</i>
<i>Figure 6.5 Examples of hand-drawn circles</i>	<i>148</i>
<i>Figure 6.6 A snapshot of training example 7 using CD</i>	<i>150</i>
<i>Figure 6.7 A snapshot of training example 7 using NL</i>	<i>151</i>
<i>Figure 6.8 A snapshot of “UpdatePatientRecord” Event (Question Part-A) using CD</i>	<i>152</i>
<i>Figure 6.9 A snapshot of “UpdatePatientRecord” Event (Question Part-A) using NL</i>	<i>153</i>
<i>Figure 6.10 A snapshot of “UpdatePatientRecord” Event (Question Part-B) using CD</i>	<i>154</i>
<i>Figure 6.11 A snapshot of “UpdatePatientRecord” Event (Question Part-B) using NL</i>	<i>155</i>
<i>Figure 6.12 A snapshot of the correct answer of “UpdatePatientRecord” Event Question using CD</i>	<i>156</i>
<i>Figure 6.13 A snapshot of the correct answer of “UpdatePatientRecord” Event Question using NL</i>	<i>156</i>
<i>Figure 6.14 Graph of the average of correct answers for the two groups across the 21 questions</i>	<i>158</i>
<i>Figure 6.15 Graph of the interaction of CD and NL representations and correct answers of the two parts of the 21 questions</i>	<i>158</i>
<i>Figure 6.16 Graph of the average of correct objects for the two groups across the 21 questions</i>	<i>160</i>
<i>Figure 6.17 Graph of the interaction of CD and NL representations and correct objects of the two parts of the 21 questions</i>	<i>161</i>
<i>Figure 6.18 Graph of the average of created objects for the two groups across the 21 questions</i>	<i>162</i>
<i>Figure 6.19 Graph of the interaction of CD and NL representations and created objects of the two parts of the 21 questions</i>	<i>163</i>
<i>Figure 6.20 Two disjointed sets</i>	<i>165</i>
<i>Figure 6.21 Two sets with the same size and xy points</i>	<i>165</i>
<i>Figure 6.22 Two related sets</i>	<i>165</i>
<i>Figure 6.23 Graph of the average of configuration for the two groups across the 21 questions</i>	<i>166</i>
<i>Figure 6.24 Graph of the interaction of CD and NL representations and configuration of the two parts of the 21 questions</i>	<i>167</i>
<i>Figure 6.25 Graph of the average of steps for the two groups across the 21 questions</i>	<i>168</i>
<i>Figure 6.26 Graph of the interaction of CD and NL representations and steps of the two parts of the 21 questions</i>	<i>169</i>
<i>Figure 6.27 Graph of the average of returns to examples for the two groups across the 21 questions</i>	<i>171</i>
<i>Figure 6.28 Graph of the interaction of CD and NL representations and steps of the two parts of the 21 questions</i>	<i>172</i>

<i>Figure 6.29 Graph of the average of time spent on each example for the two groups across the 7 examples.....</i>	<i>173</i>
<i>Figure 6.30 Graph of the average of time spent on each new example for the two groups across the 7 examples.....</i>	<i>174</i>
<i>Figure 6.31 Graph of the average of time spent on each returned example for the two groups across the 21 questions.....</i>	<i>175</i>
<i>Figure 6.32 Graph of the average of time spent on each returned example for the two groups across the 7 examples.....</i>	<i>175</i>
<i>Figure 6.33 Graph of the interaction of CD and NL representations and the returned examples time of the two parts of the 21 questions.....</i>	<i>176</i>
<i>Figure 6.34 Graph of the average of time spent on each initial thinking for the two groups across the 21 questions.....</i>	<i>177</i>
<i>Figure 6.35 Graph of the interaction of CD and NL representations and initial thinking time of the two parts of the 21 questions</i>	<i>178</i>

Chapter 1 Introduction

This chapter has four sections: Section 1.1 presents the reasons leading to this study, section 1.2 sets out the goals and objectives, section 1.3 defines the methodology we adopt and finally section 1.4 is an overview of the chapters that follow.

1.1. Motivation and Background

Program specification is a description of the software system that must be available to software designers in advance to eliminate guesswork and to understand the limitations of the proposed system, to ensure that all domain states are represented. It is either formal, if mathematical notations are used, or informal, if narrative descriptions are used. The struggle to choose formal or informal languages is due to the involvement of stakeholders, who are experts in a domain, and software developers, who are experts in program specification in the development of software systems, which requires sufficient shared understanding of the proposed design's representation. Unlike software designers, stakeholders are experts in their own domain but they are novices in designing systems. On the one hand, formal specification languages, which ensure unambiguous semantics and offer a single interpretation, can be used to represent a design. However, the problem with using these formal languages is that they are used as a *lingua franca* by experienced software engineers to communicate between themselves. They require specialist knowledge, good experience and a strong mathematical background. Thus, it is difficult to engage stakeholders in conversations about the design of the model. On the other hand, most of the time informal specification languages, which are easy to use, have the problem of ambiguity and they allow different inferences. As a result, the initial step in designing a solution to a problem is to choose an appropriate representation.

Constraint diagrams (CD) (Kent, 1997) are a diagrammatic formal program specification language, which are used to model programs in the preliminary stages of designing complex software systems. The notion of this proposal is to bridge the gap between formal and informal specification languages by providing an intuitive diagrammatic formal language that is simpler and more effective than other approaches used to formally specify programs. Moreover, CD is for novices in specific areas, either in domain or program specification. Although CD is a simple diagrammatic language, it

can specify functions or relations, show the properties of those relationships and compositions of those relationships, express constraints, and enhance the visualization of object structures. The CD notation is used to alternate the mathematical logic notation, to express constraints in Syntropy (Cook & Daniels, 1994), Catalysis (D'Souza & Wills, 1995; D'Souza & Wills, 1998), Z (Woodcock & Davies, 1996) and Object Constraint Language (OCL) (Warmer & Kleppe, 1998). Not only that, but also it is used in conjunction with the Unified Modelling Language (UML) diagrams (Booch, et al., 1999; Oestereich, 2002) that specify large software systems, due to CD's ability to visualize the properties of relationships and the relative positions of the elements such as being a subset of other sets. Constraints (Burns & Hajdukiewicz, 2004), which are a set of important relationships that affect the accuracy of the system, are visible only when they are represented and understood. Figure 1.1 provides an example of a constraint diagram taken from a health informatics case study called a Patient Record System (Fetis, et al., 2005). Although superficially it might seem visually straightforward and simple, there is much in the structure of the diagram. Each contour is used to represent a set of elements, a labelled arrow to represent a binary relation, a dot to represent an existential element (existential spider), and an asterisk to represent a universal spider. This diagram shows that for each Health Professional (*HProf*), the Patient Record (*PRec*) of any *patient* who is related to the *HProf* must be originated by all *services* which are associated to the *HProf*. Clearly there is a level of conceptual complexity here, even though the diagram may visually be relatively simple. Reading CD notation can cause ambiguity if it does not adopt a reading choice. For Figure 1.1, we read this diagram using the 'follow the arrow' convention. However, we can use other implicit or explicit reading choices such as an explicit reading tree (Fish, et al., 2003; Fish & Howse, 2003; Fish & Howse, 2004; Fish & Masthoff, 2005) which eliminates the intuitive reading possibilities, or the implicit reading tree (Howse & Schuman, 2005) which provides a simple default reading option.

This research aims to explore the usability of CD which has not yet been evaluated. The usability, according to ISO 9241-11, is "*the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use.*" (Bevan, 1997). CD needs to be evaluated to test its usability, which according to the previous definition, checks its ability to be used by novices, to specify programs, to examine its real impact in the real world.

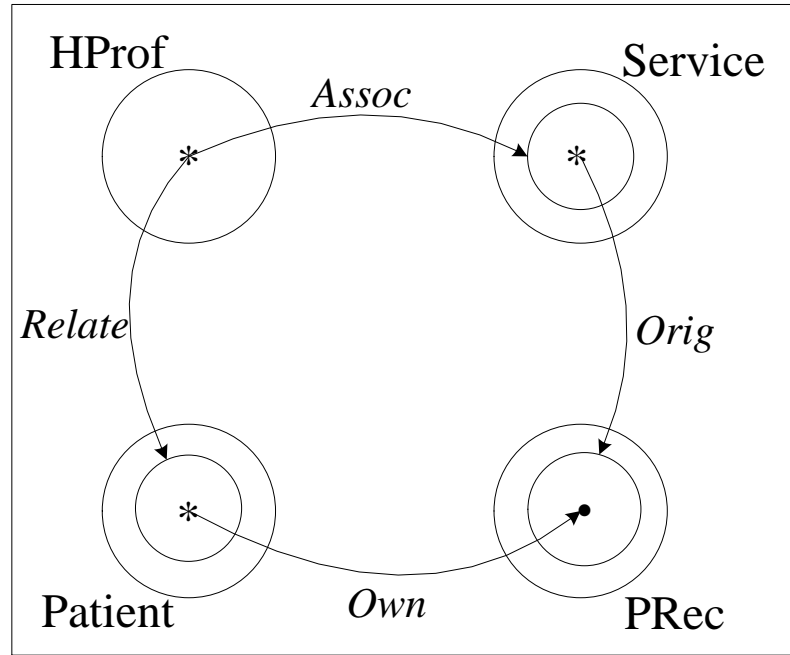


Figure 1.1 Constraint Diagrams representation of the invariant of the Patient Record System

This thesis includes theoretical and empirical investigations to evaluate the usability of CD as a visual modelling language for diagrammatic communication between involvers in the program designs. Since CD is a language, we will start with a definition of the language: a language is a tool that we use for interactions. A language is defined as “*a system of symbols and rules that enable us to communicate*” (Harley, 2008). According to Whorf’s hypothesis, in the weak interpretation (Carroll, 1956), the structure of a language affects the thinking habits and the behaviours of the human. Thus, this applies to the language that we might consider to express program specifications. So, it is important to pick a good language. With different languages there are different conceptual costs and benefits which need to be evaluated.

Since some basis of comparison is needed in order to evaluate the potential of CD, several formal or informal, diagrammatic or non-diagrammatic languages are likely to be used and compared with CD. Examples of these languages are: the first-order logic (Margaris, 1990), Z (Woodcock & Davies, 1996), and the UML (Booch, et al., 1999). Although the idea of CD notation is to try to bridge such a gap by providing an intuitive formal language with precise semantics, due to their diagrammatic properties such as being well matched to its meaning (Gurr, 1996) which shown in Figure 4.19 and having free rides (Shimojima, 1996) which is shown in Figure 2.4 – as will be discussed extensively with examples in chapter 4 – it is used as an aid to communicate with all

stakeholders as in section 3.6, even those not mathematically trained. For this reason, it is unlikely that novice users with little technical knowledge of program specification would use them. Therefore, for the purpose of this experiment, natural language (NL) provides a realistic and suitable comparator. Although some may question the legitimacy of using NL as the comparator in this study instead of formal languages, we believe NL has three advantages over the mentioned ones. First, it is a common notation that is readily understood. Second, it is not demanding or beyond novice users' capabilities, which is necessary for communicating requirements between the client and the software engineers. Third, it is always used in specification as a narrative specification to produce requirements specification documents that go with the formal specification of software. Therefore, this could result in exploring answers to scientific questions about diagrams and notations.

1.2. Research Goals and Objectives

Due to the involvement of experts in a domain and experts in program specification in the development of software systems, there is a need for sufficient shared understanding of the problem's representations and of the proposed solution. For the purposes of formal software specification using a visual method, Constraint Diagrams (CD) (Kent, 1997) were proposed as a simple and effective approach to formally specify programs (Howse & Schuman, 2005). The efficacy of some aspects of CD were questioned (Stapleton & Delaney, 2008), claiming that it is not always well matched to meaning (Gurr & Turlas, 2000) and changes were suggested to improve the language. Additionally, in order to evaluate the efficacy of CD, some empirical studies have been carried out on CD. These took the form of experiments with no direct comparison to other methods (Fish & Masthoff, 2005), or a comparison of CD with VisualOCL (Bottoni, et al., 2001; Kiesner, et al., 2002) from a theoretical perspective without empirical evaluation (Fish, et al., 2005a). In this respect, no such empirical studies have yet been performed, and thus this work is a first step in this direction. In fact, the need for empirical studies to compare CD with textual languages has been pointed out (Stapleton & Delaney, 2008).

The objective of this research is to provide triangulated evidence of the potential benefits of CD notation compared to another notational system. This is important to support the hypothesis that a novice in either a domain or a program specification can understand CD notation and can actually use it to formally specify programs, which will

allow all the parties to the development of the software design to be involved without the need for a strong mathematical background that is usually required by formal specification languages. Despite the fact that CD notation is designed as an easy and simple program specification notation, we would like to raise the following questions:

1. Is CD notation effective for program specification?
2. Is CD a good notation for the construction of program specification expressions?
3. Is CD a good notation for the interpretation of program specification expressions?
4. What are the relative strengths and weaknesses of CD notation and conventional NL notation?
5. Is CD notation effective for supporting novice users?

According to these questions and their answers, this study will focus on the insights of CD notation, and will also explore the nature of diagrams that could make an effective notation, and how logic and program specification could be coded using CD notation.

1.3. Research Methodology

The overall aim of this research is to rigorously examine constraint diagrams as a software design approach, to understand the benefits and the limitations of CD. Multiple methods are adopted in order to provide triangulated evidence of the potential benefits of CD notation compared to other notational systems. Three main approaches are adopted in this research. The first is examining the semantic and task analysis of CD language. This is conducted by the application of the Cognitive Dimensions framework (Green & Petre, 1996) which is used to examine the relative strengths and weaknesses of CD and conventional notations in terms of the cognitive facilitation or impediments of these different representations which will help in determining if the users would be able to use CD to accomplish a set of tasks. The second approach is the direct empirical valuation of the constraint diagrams compared to NL in terms of the comprehension of notational system. The third approach is the another empirical valuation of the constraint diagrams compared to NL in terms of the usage of that system to generate the specifications that model the program expressions. These two empirical approaches include two computer-based experiments, one on the comprehension of a notational system and the other on the usage.

Comprehension is an important software engineering activity to facilitate reuse, inspection, and extension of existing system. There are many research communities concerned with program comprehension such as IEEE International Conferences on Program Comprehension (ICPC) and Psychology of Programming Interest Group (PPIG) annual workshops. We examined state of the art of program comprehension by using the Cognitive Dimensions of the Notations Framework and the state of the practice by conducting an experiment on the interpretations.

Construction is another important software activity to understand the usability of a notational system in real world situations.

1.4. Overview of the thesis

This section presents the overall plan of the thesis.

1.4.1. Chapter 2: A Review of the Literature

This chapter reviews the literature in six areas that relate to the aim of this study. First, we review the available program specification languages to point out their strengths and limitations. Second, we explore the diagrammatic logic and program specification languages. In this context, we compare the diagrammatic and non-diagrammatic languages for program specification. Third, we investigate the learning approaches that support logic and program specification learning activities. Fourth, this track of investigating the literature requires an understanding of the nature of the diagrammatic notations. Fifth, to concentrate on constraint diagrams, we need to explore its family by providing a review of the spider diagram family to which constraint diagrams belong. Finally, researching on the usability of a notation needs an evaluation of the available evaluation methods, highlighting the strengths and limitations of these methods and choosing the most appropriate method.

We conclude this chapter by emphasising the most relevant work related to this research.

1.4.2. Chapter 3: Constraint Diagrams

This chapter explains constraint diagrams in more detail. It shows both their formal and informal syntax and semantics. Moreover, we provide examples of the usage of constraint diagrams in program specification. Finally, we provide a comparison between

different program specification diagrams and constraint diagrams to investigate the strengths and the limitations of these related diagrams.

We provide a summary of the potential benefits of constraint diagrams to serve as the foundation of CD which will impact on understanding the next chapters.

1.4.3. Chapter 4: Cognitive Dimensions Analysis

The content of this chapter is about using a Cognitive Dimensions framework to evaluate the usability of CD and conventional notations in terms of the cognitive facilitation or impediments of these different representations. For this reason, to understand how constraint diagrams work, we must show how constraint diagrams use visual characteristics to support particular qualitative inferences. The findings of this evaluation will answer our questions as to whether CD notation cognitively works at all, whether CD is cognitively effective for supporting novices in specifying programs, and what the relative strengths and weaknesses of CD notation and conventional NL notation are in terms of the cognitive facilitation or impediments.

1.4.4. Chapter 5: Experiment 1: Interpretation of Constraint Diagrams

In this chapter, an empirical experiment to examine the interpretation of using CD compared with NL will be discussed. The findings of this evaluation will answer our questions as to whether CD notation works at all in terms of the interpretation, whether CD is a good notation for interpreting program specifications compared with NL notation, whether CD is effective for helping novices to understand programs, and what the relative strengths and weaknesses of CD notation and conventional NL notation are in terms of the comprehensive facilitation or impediments.

1.4.5. Chapter 6: Experiment 2: Construction of Constraint Diagrams

Another empirical experiment to examine the usage of CD in constructing specification compared again with NL is conducted and discussed in this chapter. The findings of this evaluation will answer our questions as to whether CD notation works at all in terms of the construction, whether CD is a good notation for construction program specifications compared with NL notation, whether CD is effective for supporting novices in constructing programs, and what the relative strengths and weaknesses of CD notation and conventional NL notation are in terms of the comprehensive facilitation or impediments.

1.4.6. Chapter 7: General Conclusions

This concluding chapter presents the findings of the triangulated evidence we adopted and discusses the contributions made by the thesis and the directions for future work. General questions will be answered.

Chapter 2 A Review of the Literature

This chapter represents the work related to this research. Indeed, this thesis draws on the literature from (a) cognitive science of representations, particularly diagrammatic representations, (b) logic and (c) program specification using diagrammatic notation.

2.1. Introduction

There is a trend to use diagrams in logic and program specification because “a picture can be worth 10,000 words” (Larkin & Simon, 1987). Hadamard (Dreyfus, 1994), in 1945, “concludes that [mathematicians], very generally, use images and that these images very often are of a geometric nature. He recounts that when thinking, practically all mathematicians avoid not only the use of words but also algebraic and other symbols; they use vague images”. With this idea in mind, an increasing number of investigations into the cognitive, logical, and computational characteristics of diagrammatic representations follow the importance of visual information in communication and computation and the key role that design, and therefore modelling notations, play in the development process of software systems.

Constraint Diagrams (Kent, 1997), as in Figure 1.1, are a language designed to formally specify information systems by visualizing logical or set-theoretic assertions and representing relationships between sets, such as containment and disjointedness. They generalize the intuitive system of Venn-Peirce diagrams investigated by Shin, by providing facilities for quantification and navigation of relations.

Another system which arises from this constraint diagrams investigation is called Spider diagrams (Gil, et al. 1999; Howse, Molina et al. 1999). Spider diagrams, as in Figure 2.6, are a system of visual notation for expressing logical statements, and form the basis of more expressive constraint diagrams. Spiders are used to denote that an element exists in a set which is the union of one or more regions, as will be shown in the next chapter. In general, spider diagrams notation is a fragment of constraint diagrams which extend spider diagrams by using additional syntax such as arrows and wildcards.

This thesis concentrates on studying the usability of Constraint Diagrams, and thus we need to understand the reasons behind (1) the need for program specifications and the difficulties associated with program specification, (2) diagrammatic representations of

logic and program specification, (3) approaches to supporting the learning of program specification, (4) the theory of diagrammatic representations in general, (5) the notion of the constraint diagram family, and (6) the different methodologies and frameworks that are used to analyse different representations. In general, this chapter will be a guide to the design of our usability study.

2.2. Program Specification Languages

There are different program specification languages which are used to provide relevant aspects of the specification. Some of them are informal such as natural languages, and the others are formal with strong mathematics such as Z language (Woodcock & Davies, 1996) and Object Constraint Language (OCL) (Warmer & Kleppe, 1998).

Figure 1.1, in Chapter 1, shows by using constraint diagrams (CD) that for each Health-professional (HProf), the Patient-Records (PRec) of any patient (Patient) who is related to the HProf must be originated by all services (Service) which are associated with the HProf. As is known, this description in natural language could result in ambiguity. Figure 1.1 could be presented in OCL as shown in Figure 2.1:

Context Patient Record System

Inv: HProf.allInstances ->forAll(h | h.Relate->forAll(p | h.Assoc->forAll(s | s.Orig->includes(p.Own))))

Figure 2.1 OCL representation of the invariant of the Patient Record System

However, this OCL does not include the disjointedness information also in the diagram e.g. HProf is disjointed from Service. Figure 2.1 shows that it will always need a mathematical background to interpret this expression. Although OCL is textual first order logic to describe additional constraints about the objects in the model, it is not a stand-alone language. All attributes used in OCL expressions must be defined in UML model. OCL expressions are verbose in that they are textual and rely on a class diagram for context. On the other hand, CD can make many statements in a single diagram and are therefore reasonably concise.

Constraints need to be described using a formal language such as OCL or Z to avoid ambiguities. OCL is a logical-based language used to define invariants and pre and post conditions of operations. Despite the fact that the problem with such languages is the

need for a strong mathematical background, and thus they are difficult to use as an aid for integrating stakeholders into the software development because they cannot read it or write it, these languages are still the only approach to verify critical systems.

In the OCL v2.3.1 manual (OMG), it is written that *“In order to write unambiguous constraints, so-called formal languages have been developed. The disadvantage of traditional formal languages is that they are usable to people with a strong mathematical background, but difficult for the average business or system modeller to use. OCL has been developed to fill this gap. It is a formal language that remains easy to read and write.”* However, Craig Larman (Larman, 2001) suggested that *“Unless there is a compelling practical reason to require people to learn and use the OCL, keep things simple and use natural language”*. In general, OCL is not a stand-alone language and needs a UML class diagram to accompany it.

After studying the advantages and disadvantages of using informal and formal languages, and for the reason of focusing on novice users, we will choose to use natural language which is informal language to be compared with CD in our research. Thus, this section guided the design of our usability study.

2.3. Logic and program specification diagrams

The development of diagrammatic notations in program specifications enables more people to accomplish more ambitious tasks. There was a tremendous development in diagrammatic representations to specify the requirements to build a program after the 1940s due to the development of computer systems and the need for presenting a design of the software.

Examples used for program specifications are Syntropy (Cook & Daniels, 1994), Catalysis (D'Souza & Wills, 1998), Entity-Relationship Diagrams (ER-D) (Chen, 1976; Chen, 2002), Harel statecharts (Harel, 1987), and The Unified Modelling Language (UML) (Booch, et al., 1999). ER-D represents data in an abstract conceptual manner that describes a database. As shown in Figure 2.2, entities are represented by rectangular boxes which are stored in tables such as *Patient* and *Patient-Record* entities. Relationships are represented by diamond-shaped boxes such as *Own*. Entities and relationships are mapped to their value sets by attributes that are depicted by circles, such as *Name*, *ID*, and *Address* for *Patient* and *Author*, *Origin*, and *Date* for *Patient-Record* as shown in the figure. So we say that a *patient* whose name, ID, and address are

equal to *Name*, *ID* and *Address* owns the *Patient-Record* that has the author, origin and date of *Author*, *Origin* and *Date*. However, it represents only the structure of a system without depicting any dynamic behaviour.

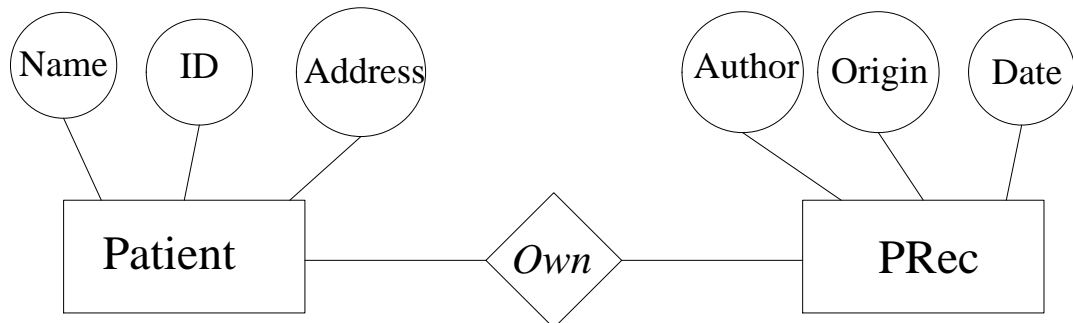


Figure 2.2 Entity-Relationship Diagram

Harel statecharts are a higraph-based system (Section 3.8.2) where the rounded rectangles (called blobs) represent states, and arrows represent transitions. They represent the dynamic behaviour of the system, and thus they allow elements to exist in multiple different states simultaneously. It has been stated that “*Modellers can use orthogonal states to decompose large state spaces naturally into independent (or almost independent) parts*” (Harel & Gery, 1997). Although Harel statecharts are used in UML as a state diagram, they are, as Harel stated, “*not exclusively visual/diagrammatic. Their non-visual parts include, for example, the events that cause transitions, the conditions that guard against taking transitions and actions that are to be carried out when a transition is taken*” (Harel, 2007). Each set represented in Figure 1.1 could be represented using different statecharts. Figure 2.3 represents the states of *Patient* set (Figure 1.1) where they could be *Alive* or *Dead*.

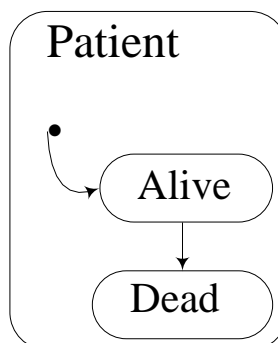


Figure 2.3 Harel's Statechart

UML v2.0 (Booch, et al., 1999), which has 13 different diagrammatic notations to provide relevant aspects of the specification, is a widespread set of different notations that has been standardized by OMG to be used in both industry and academia. Unified Modelling Language (UML) Diagrams – which includes Class Diagrams as in Figure 3.27 and includes state diagrams as in Figure 3.28 – are used to provide a ready-to-use, expressive visual modelling language (Booch, et al., 1999). It is mainly intended to be used by software designers during the software development process, from capturing the domain requirements to the implementation. It is used to simply explain your design and to give a general roadmap for the implementation. Booch, a co-creator of UML, said that the original vision for UML was a “*graphical language to help reason about the design of a system as it unfolds*” (O’Brien & Booch, 2009). By implication, unfortunately, UML is a detailed planning method with multi-stage diagrams where some diagrams are related to a specific stage or phase of the software development (e.g. UML Structure Diagrams relate to the planning and design stage because they show the real-world concepts and the relationships between them whereas the UML Deployment Diagrams relate to the installation stage because they show the way the application will be configured and deployed). Thus, UML is a good method only if it is well understood, but due to its complexity and being a method not a methodology, it could cause some issues such as confusing designers in knowing where to start. Moreover, using UML diagrams definitely requires too much time and, unfortunately, could turn the stakeholders’ focus from the design itself to the software features, which is, from the point of view of the software designers, a real interruption.

This section provided us with some of the program specification diagrams which helped us to understand the issues with them and why a new diagrams notation such as CD were proposed. Thus, this section guided us in the design of our usability study.

2.4. Approaches to supporting the learning of Logic

There are popular approaches to teaching logic (Goldson, et al., 1993), each trying to overcome the misconception about people learning logic. Barwise and Etchemendy developed a theory of heterogeneous reasoning (Barwise & Etchemendy, 1995) which was implemented as a computer program called Hyperproof for teaching first-order logic in a novel way that uses both graphical and linguistic representations, supports reasoning in either representation, or a mixture of both. The Hyperproof interface shows a grid with objects and has two main parts. One represents a diagrammatical view of the

represented first-order logic system by using different-sized objects and the other represents a list of sentences of the same system using formulas related to these objects. Then they designed Tarski's World which can be used to explore the semantics of first-order logic by writing sentences in an interpreted language, and building worlds in which those sentences may be evaluated for truth (Stenning, 2002). According to Cheng, the invention of novel diagrammatic systems can enhance conceptual learning in science and mathematics (Cheng, 2002; Cheng, 2003; Cheng & Shipstone, 2003). Diagrams could be used to make a discovery (Cheng & Simon, 1992), which would be easier than using an algebra-like representation such as those adopted by the classic models in the area of computation scientific discovery (Langley, et al., 1987). Further, by giving the same Law Encoding Diagrams (LEDs) to students of physics, learners gained similar benefits, analogous to those achieved by the original scientists, compared with learners using a conventional algebraic approach (Cheng, 1996). The Cognitive Impacts of diagrams on learning and the inferences of the relation between the represented real-world and the representing system are found in (Palmer, 1978; Barwise & Etchemendy, 1995; Zhang, 1996).

There are other approaches such as problem-solving techniques. There are ten problem-solving events such as explanation-based learning, and similarity-based learning (VanLehn, 1989). Learning gains can be measured (VanLehn, et al., 2011) by: (1) fixed tasks design, (2) fixed time design and (3) mastery learning design. A study on problem solving with diagrammatic representation came up with an interactive learning environment to examine reasoning with self-constructed diagrams (Cox, 1997), and learning-based systems must have complex statistical techniques to map between different moves of the tutor and the learner and the states of that learner (Soller & Lesgold, 2003). Taylor and his colleague (Taylor & Dionne, 2000) believe that the efficiency of solving problems relates to the level of expertise of the problem-solver in a certain domain. They believe that developing problem-solving abilities depends on the availability of problem-solving strategies. Moreover, the problem-solver's performance in terms of accuracy and time spent on solving the problem depends on being expert or novice in a certain domain (Larkin, et al., 1980). They identified the work technique adopted by novices as a backward working technique and the one adopted by experts as a forward working technique. However, experts adopt the backward working technique only on an easy problem that doesn't need any planning. Langley and Rogers believe

that if the problem-solver becomes an expert, then they will adopt the forward working technique which means a direct solution (Langley & Rogers, 2005). They state that complex tasks need to be solved with a forward working technique because working backward from the goal is unreasonable.

Larkin and Simon (Larkin & Simon, 1987) claimed that diagrams as a representation are better to use for reasoning and they rely on geometry as evidence. They showed that different representations may be better for different problems depending on the cognitive processing costs of locating the components of information. Langley and Rogers also believe that due to the visual nature of the diagrams, they have benefits for problem solving (Langley & Rogers, 2005). Indeed, after representing the problem, the problem-solver will solve the solution by adopting a strategy. Due to the cognitive and semantic properties of the diagrams, they are useful aids (Cox & Brna, 1995; Stenning & Oberlander, 1995; Cox, 1996). Moreover, computational thinking (Wing, 2006) is a fundamental skill involving solving problems, designing systems and understanding behaviours. Choosing a suitable representation is considered as computational thinking. In fact, diagrammatic-based knowledge organization is argued to be more beneficial (Koedinger, 1992) to the problem solver than sentential-based ones. Diagrams can lead to great insight, but also to the lack of it (Card, et al., 1999), as the example of the accident of the space shuttle *Challenger* showed (Tufte, 1997). This example showed that the same problem presented by different representations can have different stories because, as in that example, it was difficult to notice any patterns of damage and the adopted representation showed that the damage was low (Nielson, et al., 1997). However, the other representation showed a clear pattern of damage (Tufte, 1997) and thus, as Tufte claimed, “*there are right ways and wrong ways to show data; there are displays that reveal the truth and displays that do not*”. More on the nature of diagrammatic representations for problem solving is in (Glasgow, et al. 1995).

By understanding the graphical effects on learning logic and how graphical representations can be used to teach logic, this prepares the basis for designing the learning environment for teaching users the CD notational system.

We needed to understand how to support learning of a new notation and thus, this section can be used to guide the design of our usability study in chapters 5 and 6.

2.5. Theory of diagrammatic representations

This section is about the formal description of the nature of diagrammatic representations. Gurr explained the nature of closeness of mapping between the world and its representation, and the variation or degree of similarity, which may vary from one representation to another (Gurr, 1996; Gurr, 1997). He explained the properties of the relation between representation and represented. He proposed that most representations are homomorphisms where the mapping between world and its representation are the same but a few are isomorphic where this mapping is one-to-one. There will be a homomorphic mapping from representation to world when every object's relation in the representation accurately relates to some objects' relation in the world. Moreover, the mapping could be isomorphic, which is a special case of homomorphism. Being isomorphic, or close to being isomorphic, representations are important for the validity and ease of interpretation and reasoning of tasks. He illustrated that applying external constraints on homomorphic representations, that are too expressive, and using secondary notations for insufficiently expressive representations, can achieve isomorphism. For example, Figure 4.17 is not a well-matched to meaning due to the difference between domain and habitat (Stapleton & Delaney, 2007) because it is non-isomorphic. In general, the importance of constraints will depend on the representational system limits. He also examined the meaning of being a homomorphism representation and the success or failure in being an isomorphism. It is well known in the literature that theories of diagrams fall into two categories. The first category is to provide a justification for diagrammatic reasoning in formal proofs (Shin, 1994; Hammer & Danner, 1996; Shimojima, 1996) by demonstrating their properties as soundness and completeness to show that diagrammatic systems have some of the desirable properties of sentential ones. The second category of diagrammatic theories is to explain the impact of graphical representations on human cognition by explaining what advantages diagrammatic representations have over other forms of representation such as a free ride (Shimojima, 1996). Figure 2.4 shows an example of giving information a free ride using the CD notation since it is a diagrammatic notation. In this Figure, the diagram gives us 'for free' the information that the patient Jean is not dead. The explanation of the syntactical components along with their semantics will be shown in Chapter 3 section 3.2 and section.3. This type of inferential advantage of diagrams has been noted by several

researchers such as: (Larkin & Simon, 1987; Barwise & Etchemendy, 1995; Stenning & Lemon, 2001; Shimojima & Katagiri, 2008).

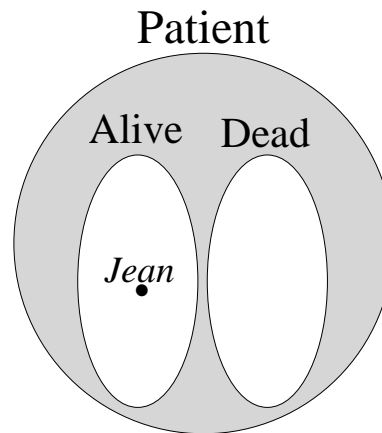


Figure 2.4 diagram to present the existence of an element in patient.

We will explain how these results were used to guide the design of our usability study in chapter 4.

2.6. The Constraint Diagram Family

The constraint-family diagrams are Euler-based diagrams which combine both the object-oriented approach and a formal method. This family consists of three diagrams: spider diagrams (Figure 2.6), constraint diagrams (Figure 1.1), and concept diagrams (Figure 2.7). For the purposes of formal software specification for using a visual method, constraint diagrams (Kent, 1997) were proposed to be used in conjunction with the UML for object-oriented modelling. It is intended to be a simple and effective approach to formally specify programs and provides a diagrammatic notation for expressing constraints (e.g., invariants) that could only be expressed textually in UML using OCL. Constraint diagrams developed due to the important role the constraints play. The designer needs to understand what the limits of the program are and to ensure that all domain states are represented. The CD notation is used to describe the syntax of a domain and to visually capture the semantics of it.

The CD was originally proposed to present a static constraint; thus, many improvements were made. For example, a three-dimensional CD (Gil & Kent, 1998) proposed for behavioural specification to express pre-conditions and post-conditions was proposed with the influence of UML multi-diagrams and Catalysis (D'Souza & Wills, 1998).

Moreover, due to the fact the CD as originally proposed are ambiguous (see Chapter 3, section 3.3.1 for an example of an ambiguous constraint diagram), Reading Tree (Fish, et al., 2003; Fish & Howse, 2003; Fish & Howse, 2004; Fish & Masthoff, 2005) was developed because some constraint diagrams have more than one intuitive reading. For explicit reading tree, the concept of the dependence graph for a constraint diagram was developed. From the dependence graph a set of reading trees can be obtained, which provides a partial ordering for some syntactic elements of the diagram to deliver a unique semantic reading. For implicit reading tree (Howse & Schuman, 2005) was proposed with the influence of Z notation (Woodcock & Davies, 1996). Furthermore, a generalized CD (Figure 4.19) was proposed (Stapleton & Delaney, 2008) to provide a sequence of reading order.

From the constraint diagram investigations by Kent and colleagues, spider diagrams (SD1) were proposed (Gil, et al., 1999; Howse, et al., 1999) as a Venn-based visual notation for expressing logical statements, and to form the basis of more expressive constraint diagrams. Later, they were extended from SD1 to SD2 to include new notation, and the inference rules were extended in order to show that the extended system is sound and complete (Howse, et al., 2000a; Howse, et al., 2001).

In general, using SD1 will prevent existential spiders from being placed in shaded zones. However, in SD2, spiders are allowed to be placed in shaded zones. Then the SD2 was extended to the ESD (Stapleton, et al., 2004a) by allowing Euler-based diagrams (rather than Venn diagrams) and by adding additional syntax such as equality. Both ESD and SD2 are similar in terms of ESD not being more expressive than SD2, and, along with further rules required for completeness, the reasoning rules for ESD are those for SD2.

Later, spider diagrams were augmented with constants (Stapleton, et al., 2004b) to contain syntactic elements analogous to constants in first order predicate logic. In general, since spider diagrams are equivalent in expressiveness to monadic first-order logic with equality (Stapleton, et al., 2004c; Stapleton, et al., 2009), they cannot be practical for being used in software modelling. Overall, constraint diagrams are more expressive than spider diagrams and they, unlike spider diagrams with equality, can express statements involving two-place predicates and are practical for being used in software modelling.

In Figure 2.6, spider diagrams are used to show the possible cases that can be expressed using the three versions of SD. By using the spider diagram (SD1), we can express that *Jean* in *Patient* and *Mrs. Peterson* in *HProf* are the same person and that this person is either a *Patient* or a *HProf*, or both as shown in Figure 2.6 part a. However, by using SD1, we cannot represent that *Jean* and *Mrs. Peterson* may be the same person and thus, SD2 is used here as in Figure 2.6 part b. Moreover, to represent that *Jean* and *Mrs. Peterson* must be the same; ESD is used as in Figure 2.6 part c.

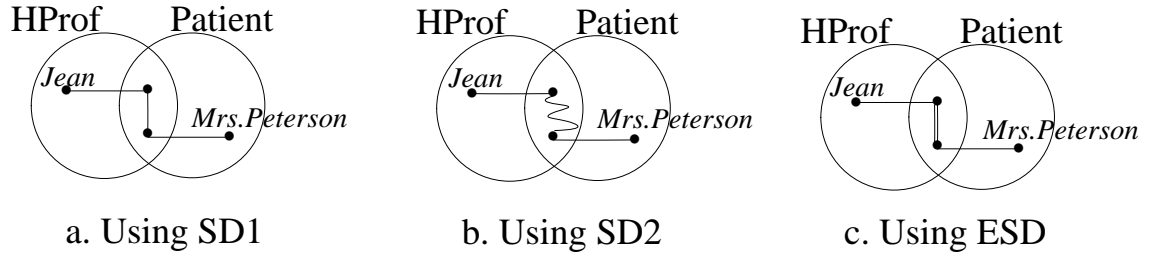


Figure 2.5 Spider Diagrams (3 versions)

Concept diagrams (Oliver, et al., 2009; Howse, et al., 2010; Howse, et al., 2011; Stapleton, et al., 2012; Stapleton, et al., 2013) are proposed to define ontology. An ontology (Gruber, 1993) is a specification of a conceptualization. They share a lot of CD syntax and augment it with new syntax. However, they have different semantics than CD in a subtle way. With the use of variables, concept diagrams do not suffer from any reading issues related to quantifiers. They are equivalent to second-order logic and thus they are more expressive than constraint diagrams, in the formal sense. Figure 2.6 is an example of a concept diagram that shows that the *Patient Jean* is the only *mother* of the *Patient John* and that *Jean* owns exactly two patient-records, both of which are *PRec*, including a *Communication* called *c*.

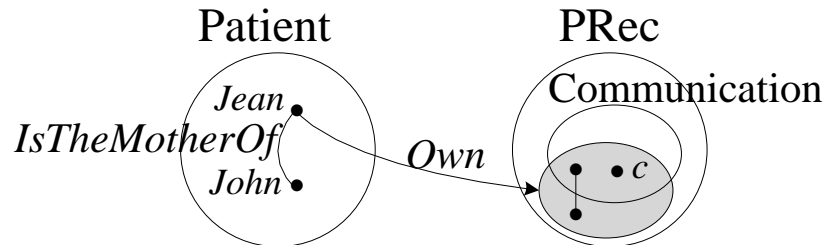


Figure 2.6 Concepts Patient, PRec, and Communication are represented by the Concept Diagrams

The semantics of concept diagrams is different from the semantics of the CD notation as discussed in (Stapleton, et al., 2013). For example, both notations have the dot which

syntactically is the same component. However, the dots in the CD notation represent quantification whereas in the concept diagrams the dots represent variables.

This section provided a broader view of the constraint diagrams family which is needed as an introduction for understanding Chapter 3.

2.7. Methodologies and frameworks for analysing notational systems

In general, there are several evaluation approaches (e.g., different aspects such as GOMS (Card, et al., 1983), Glinert's framework (Glinert, 1989), Heuristic evaluation (HE) (Nielsen & Molich, 1990; Nielsen, 1992), Maximal repeating pattern analysis (Siochi & Hix, 1991), Layout appropriateness metric (Sears, 1993), The Cognitive Walkthrough methodology (Lewis, et al., 1990; Wharton, et al., 1994), Representational Epistemological Interface Design approach (REEP) (Cheng & Barone, 2007), Ecological Interface Design (EID) framework (Vicente, 1999), Information Visualization (Card, et al., 1999), and Zhang Relational Information Displays (Zhang 1996). However, these approaches do not evaluate notational systems. To the author's knowledge, the only evaluation approach for notational systems is the Cognitive Dimensions of Notations (CogDim) framework 1996 (Green, 1989) which is a task-specific approach for analysing the usability of notational systems, user interfaces and programming languages by using dimensions which are a high-level discussion tools. Its dimensional checklist approach is used to improve different aspects of the system. Each improvement will be associated with a trade-off cost on other aspects.

CogDim is the only HCI evaluation approach intended to evaluate languages (Blackwell, et al., 2000). It concentrates on the notational design rather than the instructiveness of the user interfaces, and it is, as Green and his colleagues claimed, easy to learn and apply by non-specialists, and it is applicable at any stage of design. There are many studies that have used CogDim to evaluate the cognitive features of the languages. For example, CogDim was used to cognitively compare Prograph and LabVIEW (Green & Petre, 1996), to evaluate Pursuit (Modugno, et al., 1994), to evaluate PrologSpace (Yazdani & Ford, 1996), to evaluate design rationale representation (Shum, 1991), to evaluate continue-patterns in spreadsheets (Hendry, 1995), to evaluate modification in languages such as Basci and Prolog (Roast & Siddiqi, 1996), to evaluate domain-specific languages (Pereira, et al., 2008), and to evaluate

distributed notations (Green, et al., 2006). Microsoft used this framework as a vocabulary for evaluating the usability of their C# (Clarke, 2001), .NET development tools (Clarke, 2004), and an object oriented application programming interface (Clarke & Becker, 2003). Moreover, CogDim has been used to develop a framework called Representation Design Benchmarks (Yang, et al., 1997) which measures the static representations for a visual programming language. This evaluation method has a set of metric-based benchmarks to measure the static representations of the language.

Since CogDim is the only evaluation approach in the literature to evaluate the usability of a notational system, it will be adopted in Chapter 4 as a theoretical approach to evaluate the usability of the CD notation.

Chapter 3 Constraint Diagrams

This chapter introduces the system of constraint diagrams by giving, in section 3.1, a brief introduction of its development. We define both the formal and informal syntax of that system in Section 3.2 while its semantics are covered in section 3.3. Section 3.4 is a brief overview of constraint diagrams as a reasoning system. Section 3.5 describes the fragments of constraint diagrams. Section 3.6 illustrates the usage of constraint diagrams in program specification. Section 3.7 shows that constraint diagrams have been successfully applied to software modelling. Section 3.8 is about contrasting constraint diagrams with other diagrams that are used in program specification. Finally, section 3.9 discusses the outcomes of this chapter.

3.1. Introduction

Constraint Diagrams are Euler-based diagrams, which are a finite collection of simple closed curves (contours) to represent sets (Kent, 1997). They are proposed to visually express logical constraints on object-oriented models, and thus provide a substitute to formal methods such as Syntropy (Cook & Daniels, 1994), Catalysis (D'Souza & Wills, 1995) and OCL, the only non-visual part of the UML. Not only is CD notation a stand-alone language which has the ability to replace UML class diagrams, but also it can be integrated into several methodologies such as (Fetais, et al., 2005; Howse & Schuman, 2005) where CD is used instead of Z notations in schemas; and it fits in UML models as well. It is familiar because it is based on Euler diagrams and Venn diagrams which are widely known and it is expressive because it overcomes many of the topological restrictions of Venn diagrams such as the difficulties of drawing more than eight sets, and the cardinality. Furthermore, it is clear and unambiguous because it has formal semantics and is equivalent to first order logic, while the generalized CD (Stapleton & Delaney, 2008) is equivalent to second order logic.

CD provides a diagrammatic notation to construct an abstract information structure that is intended to specify a system by joining many statements in a single diagram, and therefore is reasonably concise, and to express static constraints (e.g. invariants) and dynamic constraints for behavioural contracts (e.g. events defined in terms of pre- and

post-conditions). The language design emphasizes scalability and expressiveness while retaining intuitiveness, and includes facilities for quantification and navigation of relations.

The purpose of this chapter is to introduce CD and to illustrate some of its characteristics. CD is a formal logic and its syntax, semantics and reasoning rules will be described in the next three sections.

3.2. Syntax

CD notation augments Euler and Venn diagrams with additional syntax, as described in (Fish, et al., 2005b; Stapleton, et al., 2005a; Stapleton & Delaney, 2008; Burton, 2011).

We will explore the syntax informally and then formally.

3.2.1. Informal syntax

This section demonstrates the comprehensiveness of CD in an easy way that people who have not had an extensive experience in formal program specification could understand.

The syntax of CD is described as follows:

- A boundary rectangle.
- A finite set of contours to represent sets. There are two types of contours: given-contours and derived-contours.
- A set of regions. A basic region is the bounded area of a contour or the boundary rectangle. Any basic region or any non-empty union, intersection, or difference of regions is a region.
- A set of zones. A zone is a single region which does not contain any other regions.
- A set of shaded zones.
- A finite set of spiders. There are two types of spiders: existential spiders (denoted by dot) and universal spiders (denoted by asterisk). A spider is a tree with nodes (feet) whose habitat is the set of zones in which the feet are placed and connected by a straight line (legs).
- A finite set of arrows. An arrow is a relation between a spider or a contour as a source and another spider or contour as a target. (Fish, et al., 2005b)
- A set of labels which can be associated with contours or dots, but always associated with arrows.

- A reading tree provides a partial ordering of the quantifiers (spiders) in the diagram and could be explicit or implicit.

In general, a reading tree ensures a unique semantic interpretation because the reading is a FOPL sentence. The explicit reading tree will help advanced users to express complex constraints with unambiguous semantics. This means the syntax of the CD notation will be extended to include a reading tree. However, this will affect new users who want to learn and use the CD notation. Overall, explicit will be used for optimizing expressiveness, but it will increase the complexity of the diagrams and the need for mental operations.

On the other hand, the implicit reading tree case will simplify the diagram syntax, which will facilitate this notation learning for new user, but will leave some complex diagrams ambiguous. Thus, to solve the ambiguity issue, as in Figure 3.1, research has been done on a method called the default readings of constraint diagrams, where no reading tree is given (Fish & Howse, 2004) by adopting a particular choice of reading tree depending only on the diagram properties to give a unique reading. The default reading requires an ordering of certain sets of spiders in a diagram for a unique semantic interpretation without the need for a user to choose a reading tree. This thesis adopts the default reading method because we are targeting inexperienced users. Examples of these diagrams are shown in Section 3.6.

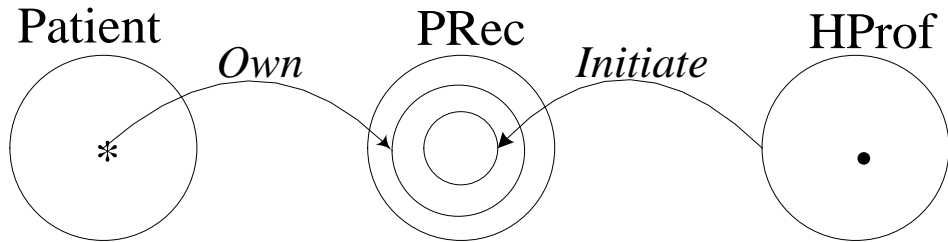


Figure 3.1 An example of an ambiguous constraint diagram

In general, the CD notation that was originally proposed is ambiguous. Figure 3.1 is an example of an ambiguous constraint diagram. In this figure there are three given contours labelled *Patient*, *PRec* and *HProf* which represent three disjointed sets. The dot is an existential spider, which represents existential quantification (there is a health professional) and its habitat is inside *HProf*. The asterisk is a universal spider, which represents universal quantification (for all patients) and its habitat is inside *Patient*. The

two arrows labelled *Own* and *Initiate* represent relations. The arrow labelled *Own* is sourced at the universal spider and targets an unlabelled derived contour inside *PRec*. This derived contour represents the image of the relation *Own* (the records which are owned). The semantics of this diagram depend upon the order in which it is read. If we start reading at the universal spider we obtain that each patient owns some records, and there is a health professional who initiates only records owned by that patient. However, if we start reading at the existential spider we get that there is a health professional who initiates some records, and each patient owns all of these records.

A constraint diagram is a single boundary rectangle which includes finite sets of given-contours, given-contour labels, shaded zones, spiders, at most one derived-contour – for a given contour – which is a target of an arrow, arrows and arrow labels.

Now let us consider the following example.

Example 3.1:

From chapter 1, Figure 1.1 has a boundary rectangle, seven contours (four given and three derived), three universal and one existential spider, four arrows, four given-contour labels and four arrow labels. The following is a description of the syntax:

Figure 1.1 shows that for each *HProf*, the *PRec* of any *Patient* who is related to the *HProf* must be originated by all *Services* which are associated to the *HProf*. To be more precise, this diagram has a boundary rectangle which represents syntactically the edge of the diagram. This rectangle contains given-contours such as *HProf*, *Service*, *Patient* and *PRec*, which are simple closed contours with labels. *HProf* is disjoint from the six other sets. All the elements of *HProf* are represented by the universal spider (asterisk) which is the source of an arrow (a binary relation) called *Assoc*. The target of this relation is a derived-contour of *Service*. So, for all *HProf*, they associate to a set of services. Another arrow called *Orig* is connecting an asterisk in a derived-contour in *Service* to another derived-contour in *PRec*. Also, *Relate* is an arrow that represents a relation which has an asterisk from *HProf* as the source and a derived-contour in *Patient* as a target. The last arrow, *Ref*, is connecting all elements of the derived-contour in *Patient* to some distinct elements (denoted by a dot) in its habitat which is a derived contour in *PRec*. Here the habitat is the same as the zone because these spiders do not have legs.

3.2.2. Formal Syntax

To make sure of the validity of the CD notation, formal syntax is described. Formal syntax is important because type-equivalence implies semantic-equivalence. A constraint diagram is formally defined as described in (Fish, et.al, 2005b) in Figure 3.2.

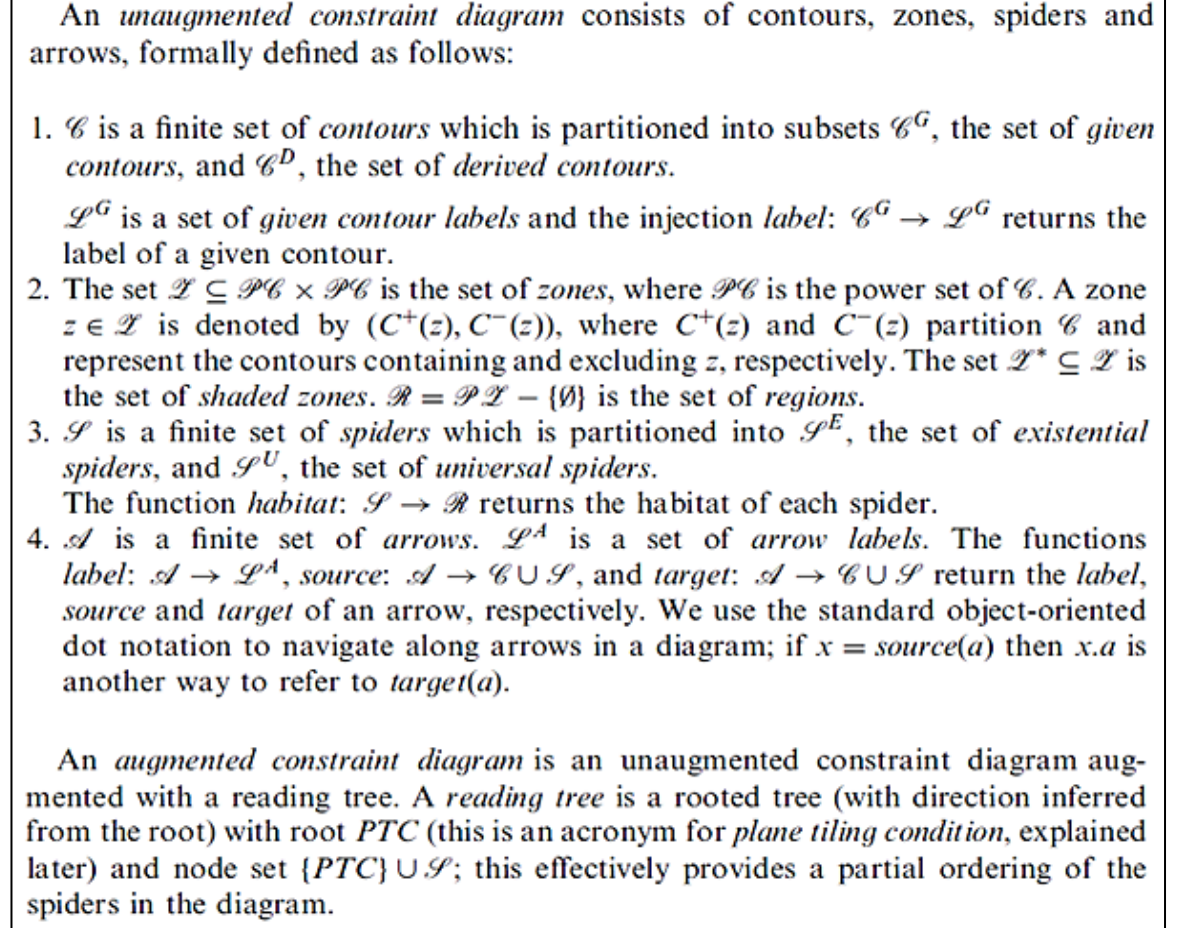


Figure 3.2 This Formal Syntax is based on (Fish, et al., 2005b)

Now let us consider the following example.

Example 3.2:

From Figure 1.1:

Taking that $\mathcal{L}^D = \{ D_S, D_{PA}, D_{PR} \}$

$\mathcal{C} = \{ HProf, Service, D_S, Patient, D_{PA}, PRec, D_{PR} \}$ where D_S, D_{PA} , and D_{PR} denotes the derived contours.

$\mathcal{L}^G = \{ HProf, Service, Patient, PRec \}$

$\mathcal{Z} = \{ (\emptyset, \{ HProf, Service, Patient, PRec, D_S, D_{PA}, D_{PR} \}) \}$,

$(\{HProf\}, \{PRec, Service, Patient, D_S, D_{PA}, D_{PR}\}),$
 $(\{Service\}, \{HProf, Patient, PRec, D_S, D_{PA}, D_{PR}\}),$
 $(\{Patient\}, \{HProf, Service, PRec, D_{PA}, D_S, D_{PR}\}),$
 $(\{PRec\}, \{HProf, Service, Patient, D_S, D_{PA}, D_{PR}\}),$
 $(\{Service, D_S\}, \{HProf, Patient, PRec, D_{PA}, D_{PR}\}),$
 $(\{Patient, D_{PA}\}, \{HProf, Service, PRec, D_S, D_{PR}\}),$
 $(\{PRec, D_{PR}\}, \{HProf, Service, Patient, D_S, D_{PA}\}) \}.$

$Z^* = \{ \}$

$R = PZ - \{ \emptyset \}$

$R^* = PZ^* - \{ \emptyset \}$

$S = \{s_1, s_2, s_3, s_4\}$ where s_1 is the spider in $HProf$, s_2 is the spider in $Service$, s_3 is the spider in $Patient$ and s_4 is the spider in $PRec$.

$A = \{Relate, Assoc, Orig, Ref\}$

$L^A = \{Relate, Assoc, Orig, Ref\}$

$text(source(a_{Relate})) = "HProf"$

$text(target(a_{Relate})) = "Patient"$

$text(source(a_{Assoc})) = "HProf"$

$text(target(a_{Assoc})) = "Service"$

$text(source(a_{Own})) = "Patient"$

$text(target(a_{Own})) = "PRec"$

$text(source(a_{Orig})) = "Service"$

$text(target(a_{Orig})) = "PRec"$

This diagram is not associated with any explicit reading tree. However, in this example we used a partial ordering of quantifiers by using an implicit reading tree to give rise to the default reading, to ensure a unique meaning.

In general, example 3.2 shows the complexity of information that the diagram in Figure 1.1 represents. This shows how intuitive the diagram represented by the CD notation (in Figure 1.1) is, compared to this example.

3.3. Semantics

We will explain this section as described in (Fish, et al., 2005b; Stapleton, et al., 2005a; Stapleton & Delaney, 2008; Burton, 2011). In general, constraint diagrams without any arrows are spider diagrams, and thus their semantics extend their underlying spider diagrams' semantics.

We will describe the informal semantics and follow it with the formal semantics.

3.3.1. Informal Semantics

The boundary rectangle represents semantically the universe that we are considering. In this rectangle there are given-contours to represent sets, and derived-contours to represent the image of a relation, with respect to the topological properties of the sets. Existential spiders represent existential quantification and universal spiders represent universal quantification, such that an independent spider represents a distinct element. An arrow represents a relation such that the relation is represented by a label; its source is a spider or a contour and its target (the relational image) is also either a spider or a contour. Regions are sets and in a shaded region all the elements are represented by spiders.

Let us consider the following example:

Example 3.3:

From Figure 1.1, the diagram contains four given contours, three derived contours, three universal spiders, one existential spider, and four arrows. The existential spider asserts the existence of at least one element in *PREc*. The universal spider in a set means all elements in that set. The arrow labelled *Assoc*, together with its source and target assert that the *Assoc*, which is a binary relation, has an image which is a subset of *Service* when the domain is an element of *HProf*. The arrow labelled *Relate*, together with its

source and target assert that the *Relate*, which is a binary relation, has an image which is a subset of *Patient* when the domain is an element of *HProf*. The arrow labelled *Orig*, together with its source and target assert that the *Orig*, which is a binary relation, has an image which is an element in *PRec* when the domain is an element of *Service*. The arrow labelled *Ref*, together with its source and target, assert that the *Ref*, which is a binary relation, has an image which is an element in *PRec* when the domain is an element of *Patient*. Finally, the four given contours represent disjoint sets.

3.3.2. Formal Semantics

For this thesis, the formal semantic for the syntax in Figure 3.2 could be found in (Fish, et al., 2005b) and thus we will not reproduce it here. However, we will give the following example where The interpretation of CD is defined by a tuple $\langle U, \Psi, \emptyset \rangle$ such that:

Table 3.1 These Formal Semantics are based on (Fish, et al., 2005b)

U is the universal set, Ψ is a function mapping given contour labels, zones and regions to subsets of U , Φ is a function mapping arrow labels to relations on U ,
--

Example 3.4:

From Figure 1.1:

$$U = \{1, 2, 3, 4\},$$

$$\Psi = \{(HProf, \{1\}), (Service, \{2\}), (Patient, \{3\}), (PRec, \{4\})\},$$

$$\Phi = \{(Assoc, \{(1, 2)\}), (Relate, \{(1, 3)\}), (Orig, \{(2, 4)\}), (Ref, \{(3, 4)\})\},$$

Augmenting the diagrams with a reading tree (effectively a partial ordering of quantifiers) ensures that each diagram has a unique semantic interpretation.

3.4. Reasoning

In this thesis we are looking at constraint diagrams from the program specifications point of view. The soundness of CD and its formal system results in a formal logic. In summary, CD is expressively equivalent to the FOPL and it has sound reasoning rules, a complete reasoning system. However, it is not a decidable system due to the existence

of the universal spiders (Stapleton, et al., 2005a). In the case of reasoning, CD has diagrammatic notation advantages (Stapleton & Delaney, 2008) such as free rides that arise from the subset relations, the placements of existential spiders and the images of relations.

In general, to be able reason between constraint diagrams, reasoning rules are needed for diagram transformations (Fish & Flower, 2005). For example, in Figure 3.3 to delete an existential spider p , three conditions must not be held. First, p must not be the source or target of any arrow. Second, the habitat of p must not include a shaded zone. Finally, a universal spider x whose domain includes a foot of the spider p ; and the p must not be ordered before the node x in the tree. In this figure, p is ordered after x in the tree, so p can be deleted. However, if the reading tree had been $p \rightarrow x \rightarrow t$ then we would not be able to delete p from the diagram because the reading here will be different since $p \neq x$ in this case and this prevents us from deleting the spider p .

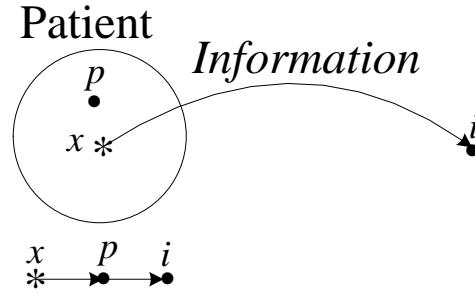


Figure 3.3 Deleting an existential spider

For more on the CD syntax, semantics, soundness, completeness and decidable system see (Gil, et al., 2001; Fish, et al., 2003; Fish & Howse, 2003; Fish & Howse 2004; Fish, et al., 2005b; Stapleton, et al., 2005a).

For CD fragments' syntax, semantics, soundness, completeness and decidable system – for Euler diagrams see (Howse, et al., 2002; Stapleton, 2005; Fish, et al., 2011); for Venn-Euler diagrams see (Gil, et al., 2002); and for spider diagrams see (Gil, et al., 1999; Howse, et al., 1999; Howse, et al., 2000a; Howse, et al., 2000b; Howse, et al., 2000c ; Howse, et al., 2001; Flower & Stapleton, 2004; Stapleton, et al., 2009).

3.5. Fragments of Constraint Diagrams

Some fragments of the CD notation, such as Hammer’s Euler diagram system (Hammer, 1995) and spider diagrams (Howse, et al., 2001; Howse, et al., 2005) have sound, complete reasoning systems and they are decidable. CD is a combination of three fragments and other syntax too. These fragments, Euler diagrams, Venn diagrams and Spider diagrams, have generalization relations: CD generalizes spider diagrams which generalize Venn diagrams, and this latter generalizes Euler diagrams. CD is more expressive than spider diagrams because of the arrows that allow expression of relationships between elements. This will be illustrated in full detail in the next chapter.

3.6. Using Constraint Diagrams in Program Specification

In this section, we show one of the uses of CD notation in the Object-Oriented Formal Specification to define invariants and to define events, as described in (Howse & Schuman, 2005). Figures 3.6 to 3.22 are redrawn from (Fetais, Howse et al. 2005) and are used as examples provided from the “Patient Record System” case study.

Figure 3.4 gives an overview of the initial stage for this development where every level represents a class, and is depicted by a rectangle with its role, and arrows indicate the extension of these levels.

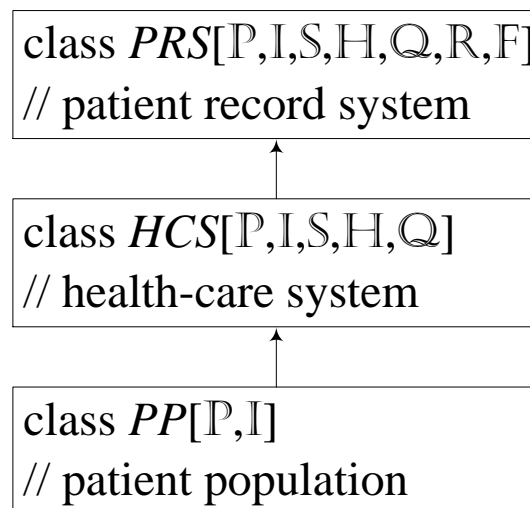


Figure 3.4 Class-structure for Patient Record System

3.6.1. Patient Population: class $PP[P;I]$

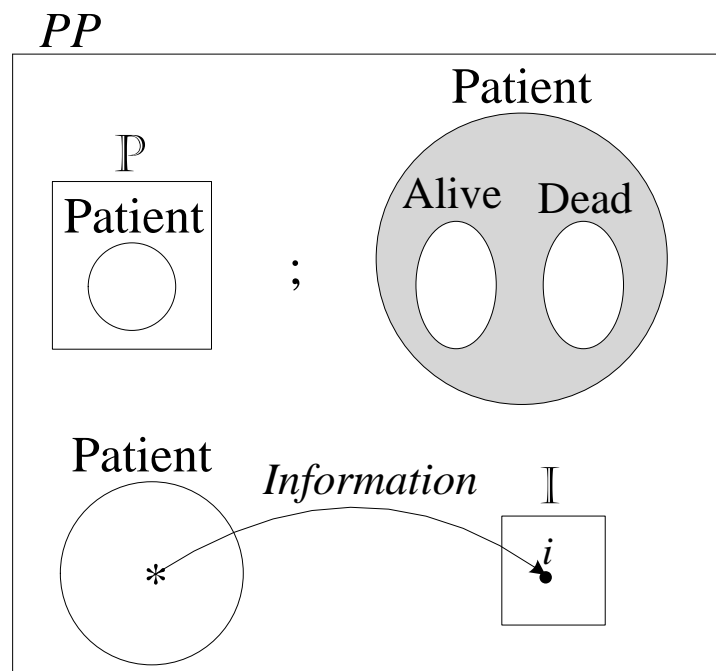


Figure 3.5 Specification of an invariant that uses constraint diagrams

Figure 3.5 provides an example of using constraint diagrams to specify an invariant (class) called PP which is taken from a health informatics case study. This invariant is about patients' personal information where a Patient Population- PP is a set of patients which is uniquely identified by elements from given type P . Each known patient has his/her own associated information of type I , and is either alive or dead. This invariant for class PP is a conjunction of three constraint diagrams: the first defines *Patient* to be a set of a given type P . The second partitions *Patient* into two disjoint subsets: those who are *Dead* and all the others who are *Alive*. The partition of *Patient* is indicated by shading the zone inside that set but outside its two subsets, identified by the names *Alive* and *Dead*. The third defines a relation *Info* that associates each element of *Patient* with some element of type I . Although superficially it might seem visually straightforward and simple, there is much in the structure of the diagram. It expresses that every known patient (uniquely identified by elements from given type P) has his/her own associated Information (of type I), and is either *Alive* or *Dead*. There are at least 20 different visual elements occurring on several nested levels. *Alive* and *Dead* subsets occur within the *Patient* set, which is of type P . A shaded zone containing no elements represents the empty set. The rectangle labelled P indicates that *Patient* is of the predefined type P . Another rectangle I represents another predefined type and yet

another *Patient* contour. Constraint diagrams use spiders to represent elements. The dot (existential spider) in *I* represents the existence of a spider (element) and an asterisk (universal spider) represents all spiders (elements) in a specific set. The arrow labelled *Info* represents a binary relation, where its source must be a spider and its target can be either an element (spider) or a set (contour). At the highest level of the diagram there is a framework, which is shown by a semi-box (open rectangle) and the class label *PP*. The semi-colon that separates declarations can easily be missed and a new line is used as another type of separator. Despite the fact that constraint diagrams notation is designed as an easy and simple program specification notation, it could be difficult to understand when presenting complex ideas.

The possible changes-of-state at this level are specified as the following three events:

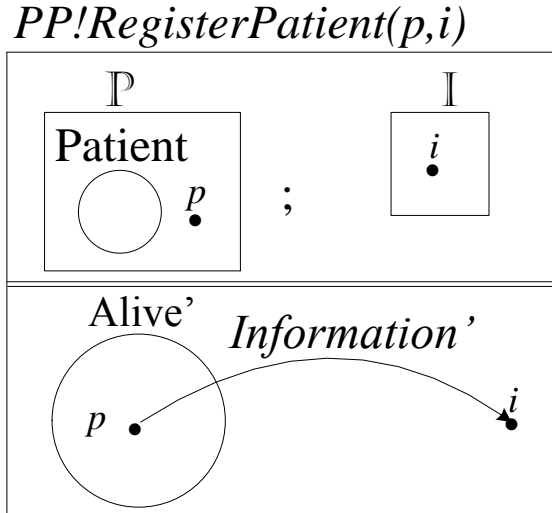


Figure 3.6 Constraint Diagrams as an Event to Register Patients

Event 1: *PP!RegisterPatient(p,i)* could be used to register a new patient – which will be identified by *p* – only if his information *i* (which is of type *I*) is given. Initially, *p* is said to be *alive* as shown in Figure 3.6. To elaborate, the CD notation can also be used to capture a more sophisticated idea: specifying an event. This Figure presents an event which is a named operation that changes an object's state by using a dash on the label to denote a change and an undashed label otherwise. Each event is specified by a pre-condition and a post-condition. The double line inside the framework is used to indicate pre-condition (above the double line) and post-condition (below the double line). The pre-condition ensures that the arguments satisfy all the constraints which are imposed by the invariant. The post-condition shows the values that are changed. However, only

minimal changes are shown because there is a convention used in constraint diagrams that "the rest stays unchanged" (Howse & Schuman, 2005). For example in Figure 3.6 in the post-condition, adding a new patient increases the *Alive*, but the *Dead* set is unchanged. As a result, the *Dead* set is not shown in the post-condition. This figure shows an event called *RegisterPatient* which is preceded by the level name and a '!' separator. This event can be used to register a new patient with information *i*, to give him/her a unique identifier *p*; initially, *p* is said to be *alive*. The pre-condition ensures that the event's argument *i* has type *I*, and *p* which is not in *Patient* yet is an identifier of type *P*. In post-condition there is a dash in the *Alive* set and another in *Information* value. In other words, the *Alive* set has increased by adding a new *patient* and the associated *Information* on that patient. These dashed names denote values that have changed.

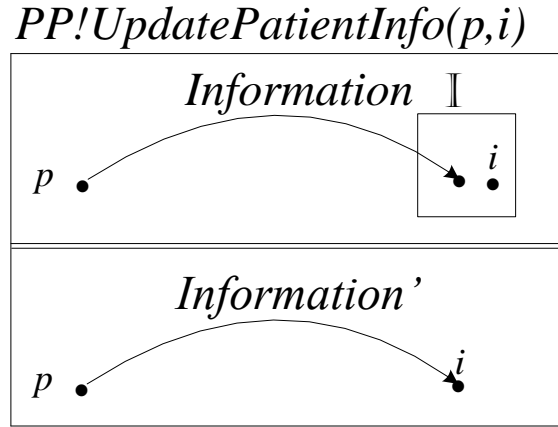


Figure 3.7 An event to update Patient Information at the PP level

Event 2: *PP!UpdatePatientInfo(p, i)* is used to update current information for patient *p*, so it has some value *i* afterwards as shown in Figure 3.7.

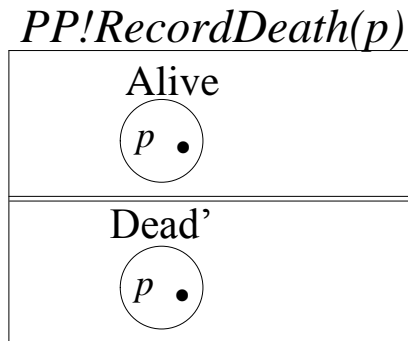


Figure 3.8 An event to record Patient's Death at the PP level.

Event 3: $PP!RecordDeath(p)$ is used to record the death of an alive patient p as shown in Figure 3.8.

3.6.2. Health-Care System: class $HCS[P;I;S;H;Q]$

Another level is developed to represent an abstract model for a Health-Care System by extending PP to define the class $HCS[P;I;S;H;Q]$.

Every Health-Care System, Figure 3.9, supports its Patient Population via some hierarchy of services and nested sub-services that each patient is currently enrolled in. Services and their subservices are uniquely identified by elements from given type S . It also maintains the central register of health-professionals (uniquely identified by elements from type H), which gives their qualifications (of type Q) and current associations with a service or sub-service; those having at least one such association are said to be active.

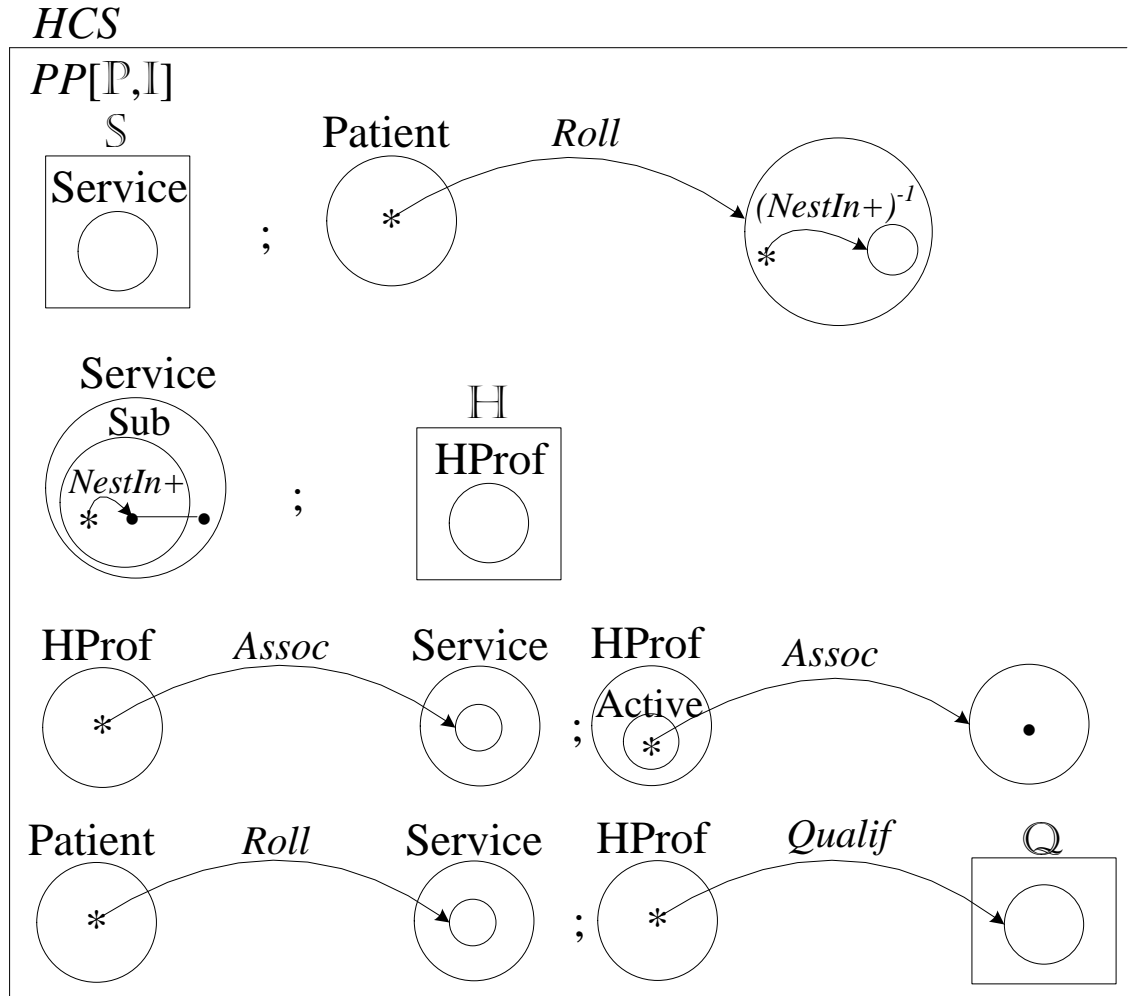


Figure 3.9 The HCS invariant

The possible changes-of-state at this level are specified as the following events:

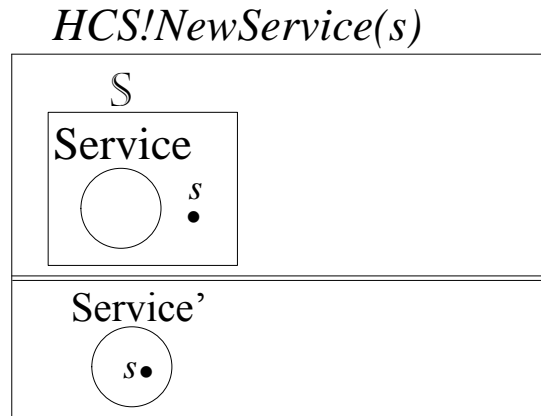


Figure 3.10 An event to define a new service

Event 1: $HCS!NewService(s)$ is used to define a new service, s , only if its name is unique as shown in Figure 3.10.

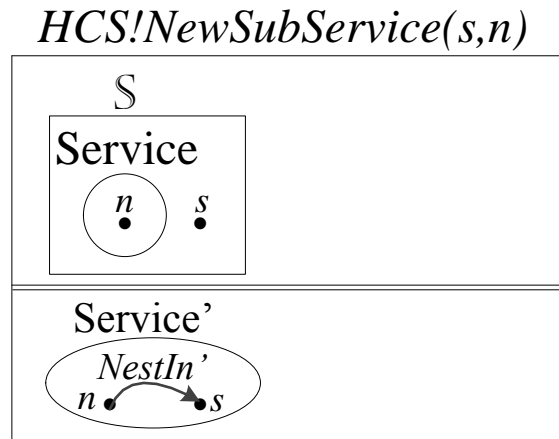


Figure 3.11 An event to define a new sub-service

Event 2: $HCS!NewSubService(s,n)$ is used to nest a new sub-service uniquely identified by s in the (sub-) service named n as shown in Figure 3.11.

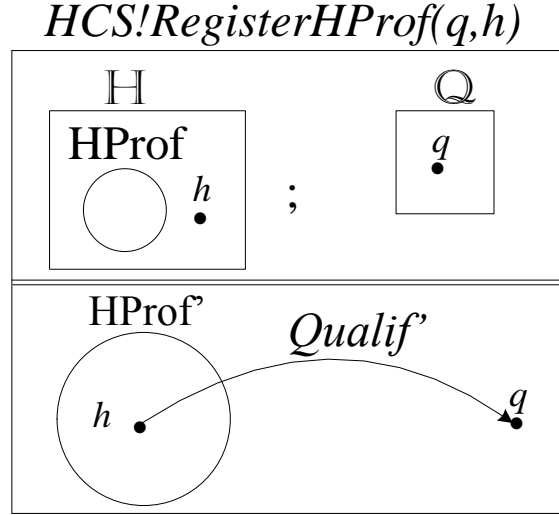


Figure 3.12 An event to register a new health professional

Event 3: *HCS!RegisterHProf(q,h)* is used to register a new professional uniquely identified by h with the qualification q , as shown in Figure 3.12.

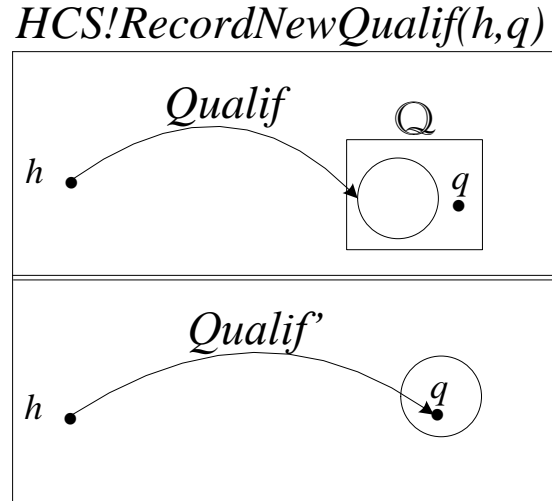


Figure 3.13 An event to record a new qualification

Event 4: *HCS!RecordNewQualif(h,q)* is used to record an additional qualification q for the professional h as in Figure 3.13.

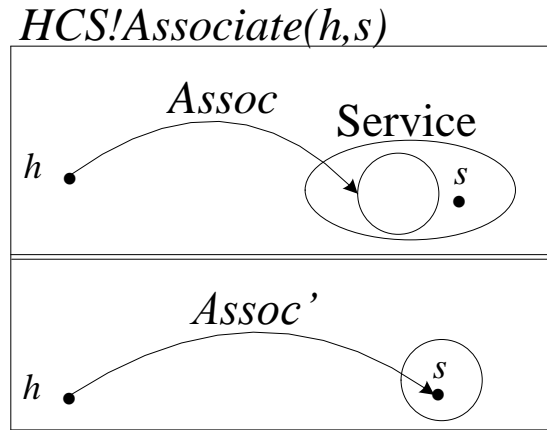


Figure 3.14 An event to associate a health-professional with a (sub-) service

Event 5: $HCS!Associate(h,s)$ is used to associate a new professional h with the (sub-) service s as shown in Figure 3.14.

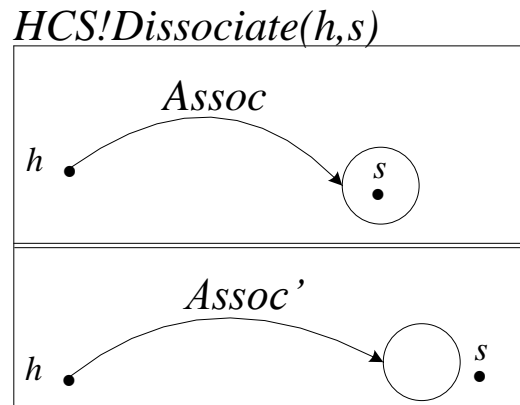


Figure 3.15 An event to dissociate a health-professional from a (sub-) service

Event 6: $HCS!Dissociate(h,s)$ is used to dissociate a professional h from the (sub-) service s as shown in Figure 3.15.

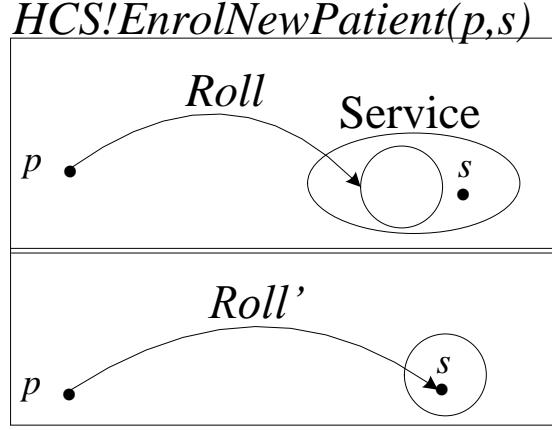


Figure 3.16 An event to enroll a patient into (sub-) service

Event 7: $HCS!EnrolNewPatient(p,s)$ is used to enrol a new patient p for the (sub-) service s as shown in Figure 3.16.

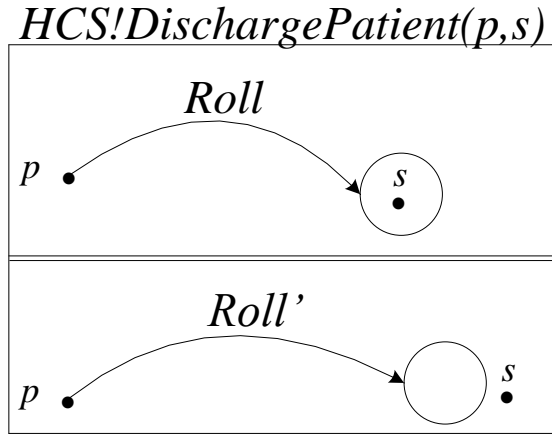


Figure 3.17 An event to discharge a patient from (sub-) service

Event 8: $HCS!DischargePatient(p,s)$ is used to discharge a patient p from the (sub-) service s as shown in Figure 3.17.

Moreover, Two PP operations are likely to be promoted, Figure 3.18, for use with the class HCS:

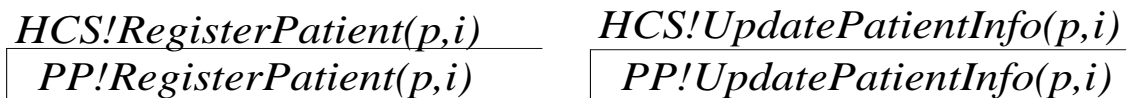


Figure 3.18 Two Events that are promoted from the extended invariant

3.6.3. Patient Record System: class $PRS[P;I;S;H;Q;R;F]$

The "Patient Record System" is a model that defines the class $PRS[P;I;S;H;Q;R;F]$ and extends class HCS .

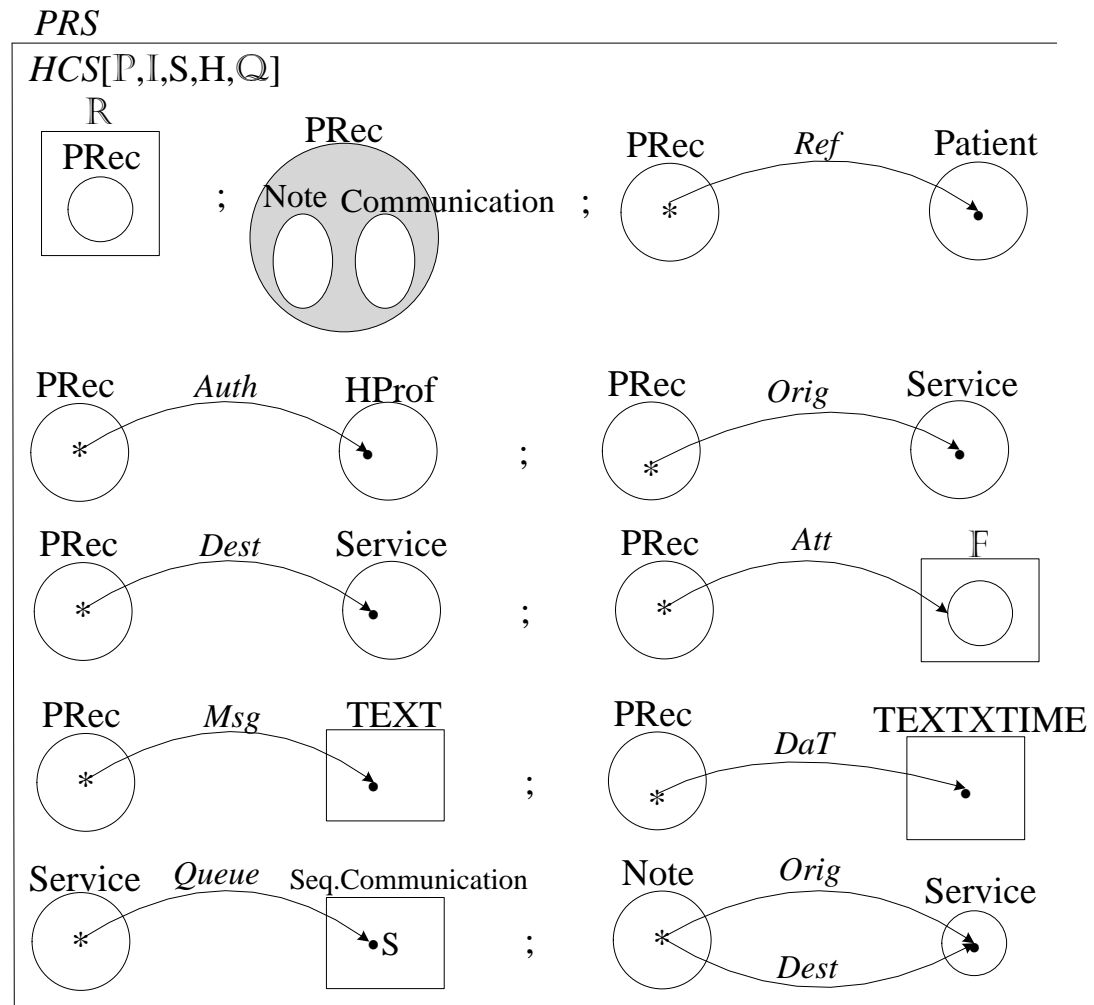


Figure 3.19 The PRS invariant

A Patient Record System, Figure 3.19, extends a Health-Care System, to support the central database of patient records (uniquely identified by elements from type R). Each such record refers to one particular patient, and identifies its author (a professional) as well as an originator and destination (services) along with a DATE and TIME of entry (introduced by the system). All have a message (of predefined type $TEXT$) and some may also have multiple attached files (of type F).

Any record having the same originator and destination is said to be a note; otherwise, it records a certain communication. These are held in a queue for their destination service.

The possible changes-of-state at this level are specified as the following events:

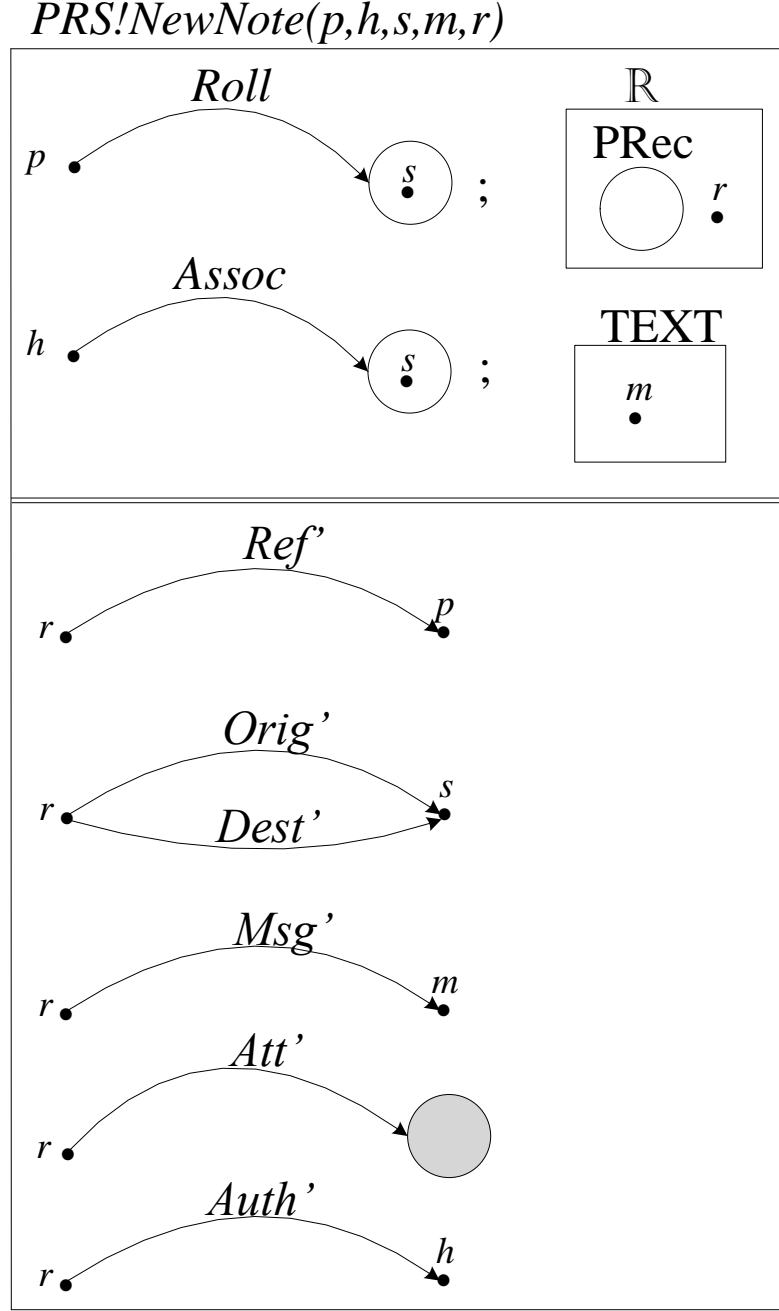


Figure 3.20 An event to create a new note

Event 1: $PRS!NewNote(p,h,s,m,r)$ is used to enter a new note – uniquely identified by r – that refers to patient p by professional h of service s with message m but no attachments, only if p is enrolled for s and h is associated with s as shown in Figure 3.20.

Event 2: $PRS!Send(p,h,s1,s2,m,F,r)$ is used to send a new communication – uniquely identified by r – that refers to patient p by professional h of service $s1$ to service $s2$ with message m and set of attached files F , only if p is enrolled for $s1$ and h is associated with $s1$. Then r would be enqueued for the destination $s2$ as shown in Figure 3.21.

$PRS!Send(p, h, s1, s2, m, F, r)$

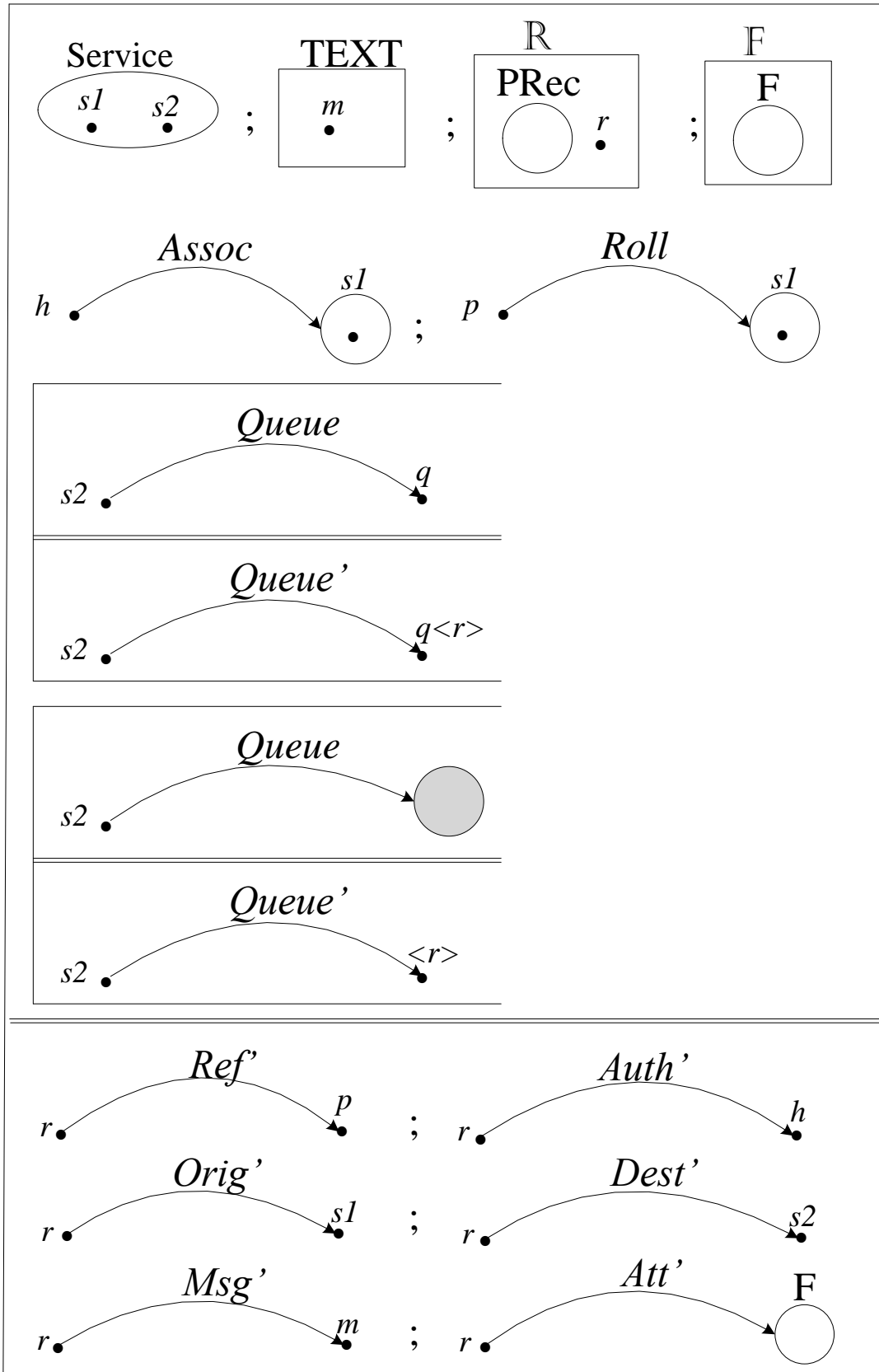


Figure 3.21 An event to send a communication

Moreover, Two HCS and one PP operations are made available for use at the level of PRS.

$PRS!Enrol(p,h,s,m,r)$	$PRS!Discharge(p,h,s,m,r)$
$PRS!NewNote(p,h,s,m,r)$	$PRS!NewNote(p,h,s,m,r)$
$HCS!EnrolNewPatient(p,s)$	$HCS!Discharge(p,s)$
$PRS!Death(p,h,s,m,r)$	$HCS!UpdatePatientInfo(p,i)$
$PRS!NewNote(p,h,s,m,r)$	$PP!UpdatePatientInfo(p,i)$
$PP!RecordDeath(p)$	

Figure 3.22 Events that are promoted from this or extended invariants

Figure 3.22 shows that at this level a record r is required to promote events such as enrol, discharge and record the death of a patient, while the update patient information is simply promoted at the level to be used at this class.

3.7. Usage

The efficacy of constraint diagrams is shown by its applications. It has been applied in many cases, for example:

- (1) A formal object-oriented specification of a software system, such as:
 - a. A library system (Kent, 1997).
 - b. A video rental store model (Howse & Schuman, 2005).
 - c. A patient record system model (Fetats, et al., 2005).
- (2) A framework for specifying an abstract model for a transparent configuration control platform for Nokia (Howse, Schuman et al., 2009).
- (3) Reasoning systems (Howse, et al., 2000c; Gil, et al., 2001; Fish, et al., 2003; Fish & Howse, 2003; Fish & Howse, 2004; Fish, et al., 2005b; Stapleton, et al., 2005a).
- (4) Developing a new diagrammatic language called Visual First Order Logic (VFOL) to visualize FOL (Stapleton, et al., 2005b).
- (5) Developing a new diagrammatic notation called Concept Diagrams to visualize ontologies (Howse, et al., 2011).

Its fragments have been used in:

- (1) Developing Euler-based diagrammatic notations such as UML diagrams and statecharts (Harel, 1987).
- (2) Presenting set-based statistical data (Chow & Ruskey, 2003).
- (3) Presenting the database complex queries' results for traditional library environments (Thièvre, et al., 2005) and for indexed video databases (Verroust & Viaud, 2004) .
- (4) Representing multi-categories non-hierarchical files systems (DeChiara, et al., 2003; DeChiara & Fish, 2007).
- (5) Representing complex genetic set relations for bio-informatics field (Kestler, et al., 2005).
- (6) Capturing knowledge in ontologies environments (Hayes, et al., 2005).
- (7) Visual editing environment for semantic web languages (Lövdahl, 2002; Zhao & Lövdahl, 2003).
- (8) Reasoning system (Shin, 1994; Hammer, 1995; Howse, et al., 2001).
- (9) Hardware specification for a safety critical environment (Clark, 2005).

These uses of CD and its fragments indicate that it is a widely accepted notation and it has been successfully used for various systems. Thus, this is why we focus on constraint diagrams in this thesis.

3.8. Comparison with other diagrams

In this section we want to compare CD with other diagrams to test the ability of these diagrams, including CD, to check which properties these diagrams are able to capture and which are absent. We know that every diagram was proposed for a different purpose. However, these diagrams shared the goal of being used in Object-oriented specification or to capture a similar kind of information. In this section we try to point out the similarities and the differences between CD and other diagrams.

3.8.1. Commutative diagram

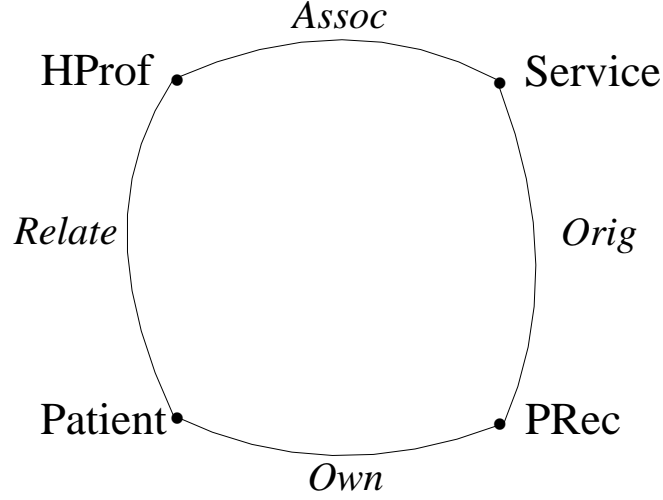


Figure 3.23 Commutative Diagrams

Commutative Diagrams (Eilenberg & MacLane, 1945) are used to describe properties of morphisms among objects such that the commutative results of all the directed paths with the same start and endpoint are equal.

Figure 1.1 represents constraint diagrams and Figure 3.23 represents commutative diagrams. Both have similar syntax and semantics, but they are not equivalent due to the differences in the structure involved. In the constraint diagrams *Assoc*, *Relate*, *Orig* and *Ref* are relations, while in the commutative diagram they are all functions. Constraint diagrams can be considered as a generalization of commutative diagrams because they produce the same accessibility by allowing relational navigation, but CD augments commutative diagrams with the set-theoretical relationships.

The semantics of the diagram presented in Figure 3.23 can be presented as:

$$\text{Orig} \circ \text{Assoc} = \text{Ref} \circ \text{Relate}$$

which means $\forall x \in \text{HProf}, \text{Orig}(\text{Assoc}(x)) = \text{Ref}(\text{Relate}(x))$

While Figure 1.1 is representing the following:

$$\forall x \in \text{HProf}, \text{Assoc}(x) \subseteq \text{Service} \text{ and } \text{Relate}(x) \subseteq \text{Patient}$$

$$\forall y \in \text{Assoc}(x), \text{Orig}(y) \subseteq \text{PRec}$$

$$\forall z \in \text{Relate}(x), \text{Ref}(z) = \text{PRec}$$

We can see that CD and commutative diagrams have similarities in both syntax and semantics. However, the structure is different because unlike commutative diagrams, CD represents set-theoretical relationships like disjointedness and subset.

3.8.2. Higraph

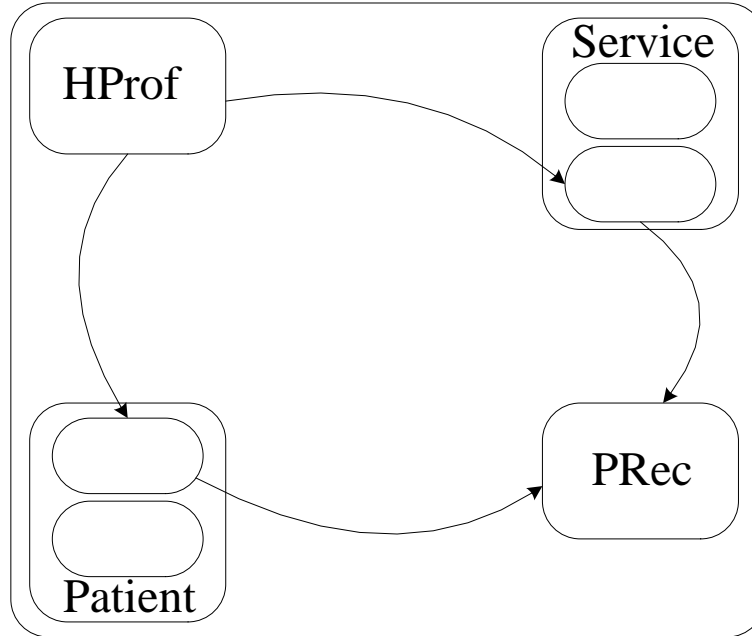


Figure 3.24 Higraph

Hi-graphs (Hammer, 1995; Harel, 1988) add arrows to Euler diagrams to represent relations between sets (Fish, et al., 2005b). Syntactically Harel's higraphs (Harel, 1988), which are the basis of Harel's statecharts (Harel, 1987), and CD have similarities such as representing binary relations between contours and being Euler-based diagrams. However, their semantics are different (Gil, et al., 2001).

Higraphs represent topological structure of dynamic behaviour. However, unlike CD, higraphs have unlabelled relations and the contours, which are called blobs, cannot represent quantifiers. Moreover, higraphs have difficulties in representing both inclusion and membership of sets (Gil & Kent, 1998). Figure 3.24 represents four sets and their binary relations. However, unlike CD, the subsets in *Service* and *Patient* are increased by one because in higraph formalism, one subset inside a set means the set itself which is not partitioning the set. So, to indicate that a set *Patient* has a subset, higraph represents this set with two internal contours to represent the area inside the subset and the other contour represents the area outside the subset. For example, in Figure 3.25 the interaction of two curves does not mean any intersection unless internal

blobs appear in it. If *Patient* had been entirely enclosed within *HProf* or vice versa, then the interpretation would be entirely different from in Figure 3.25.

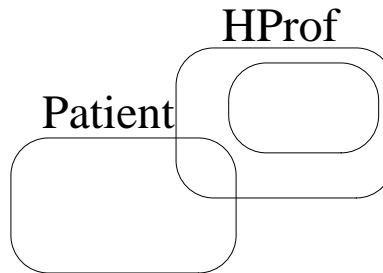


Figure 3.25 Two non-intersected blobs

3.8.3. UML diagrams

3.8.3.1. Class diagrams

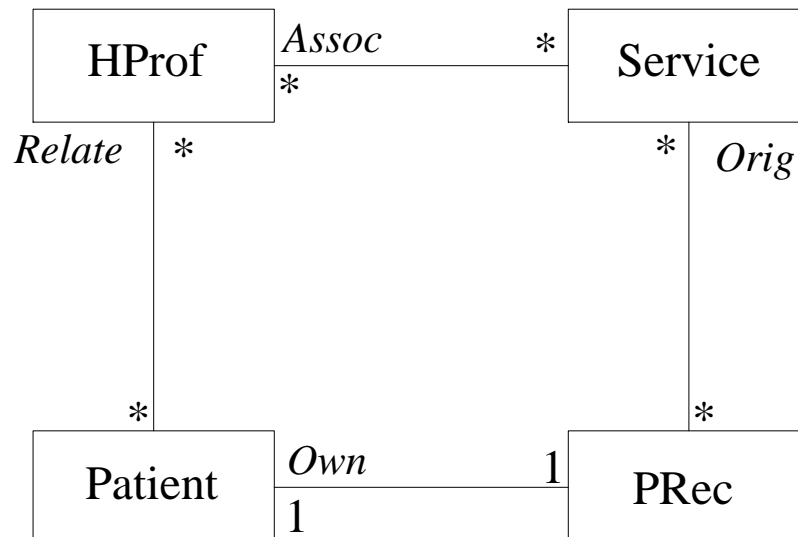


Figure 3.26 Class diagram

Class (type) diagrams are used to represent the abstract structure of the design. They express static constraints and they allow expressing the relation between classes by directed or undirected links. In Figure 3.26, we cannot tell which class has a subclass and the links are not directed, which allows reading from both directions.

CD and class diagrams represent classes, relations and cardinalities. However, CD is more expressive and they represent cardinalities visually by shadings and spiders, directed relations which show domain and range, and subsets which represent a state. Class diagrams do not show the relations within the subsets and they do not show the dynamic constraints which are expressed textually using Object Constraint Language

(OCL) which is part of UML. Unlike class diagrams, CD in Figure 1.1 shows the directed-relation navigation because we know the source and target of all relations. For example we know that the source of *Assoc* relation is *HProf* and its target is *Service*.

3.8.3.2. State diagrams

State diagrams which are based on Harel's statecharts (Harel, 1987) are used to specify dynamic behaviour. However, they are limited in the constraints they are able to express.

In Figure 3.26 we cannot tell if *Service* has subsets. If we want to represent that each *HProf* associated to a subset of *Services* we will need another diagram along with it. In this case state diagrams are used.

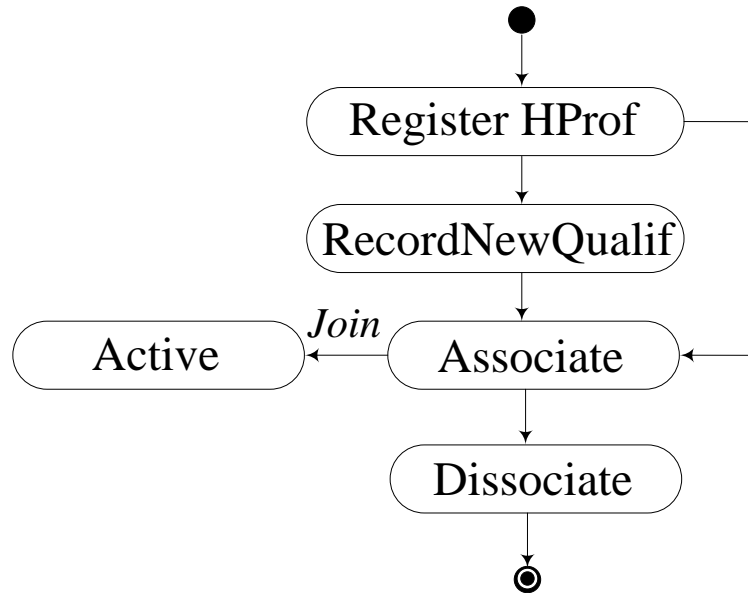


Figure 3.27 State diagram for HProf

In Figure 3.27, a *HProf* is inactive unless joining a service by an event called *Join*, which will change the state to active. This could simply be represented in Figure 1.1 by adding a subset to *HProf* and calling it *Active* and changing the source of the *Assoc* to be all elements of *Active* instead of all elements in *HProf*.

3.9. Discussion

This chapter introduces constraint diagrams by illustrating both the formal and informal syntax of that system, along with its semantics. We also gave a brief overview of constraint diagrams as a reasoning system. The fragments of constraint diagrams and

the use of constraint diagrams and its fragments in different systems ensure the ability of this notation to work successfully.

Constraint diagrams are intuitive and can be used as a stand-alone language which has the ability to replace other languages such as OCL and Z, and diagrams such as UML-class diagrams. Being expressive Euler-based diagrams is an advantage. Constraint diagrams are clearly useful for program specification because they are a formal system. In contrast to other diagrammatic notations, they are a powerful tool for program specification. Constraint diagrams seem quite complex and, thus, this is why it is an important question to see whether it is effective for program specification. Clearly there is a level of conceptual complexity here, even though the diagram may visually be relatively simple. Therefore, despite the claims about the efficacy of constraint diagrams, we might expect a user without substantial experience of the notation to find it difficult to use. However, we will start our study by answering the question of how constraint diagrams use visual characteristics to support particular qualitative inferences.

Chapter 4 Cognitive Dimensions of Notations Framework Analysis

This chapter provides a theoretical evaluation by using an existing framework to examine the usability of specific properties of the components of CD, and to analyse the cognitive tasks employing CD. Section 4.1 introduces the framework and shows its existing applications. Section 4.2 represents the activities that a notational system is desired to support, and section 4.3 presents the cognitive dimensions which are required for the activities. In section 4.4, a selection of examples from the Patient Record System is presented. In section 4.5, CD profiles are generated by applying cognitive dimensions to the defined activities. Finally, section 4.6 consists of a discussion of the application of the cognitive dimensions of notations framework to both the CD and NL notations.

4.1. Introduction

The Cognitive Dimensions (CogDim) of Notations (Green & Petre, 1996) is a heuristic framework created by Thomas Green for analyzing the usability of notational systems. Its dimensional checklist approach is used to improve different aspects of the system with a trade-off cost on other aspects.

Although there is a study on the usability of the constraint diagrams using CogDim (Morgan, 2011), this study did not provide a full usability profile of the notation. Moreover, it adopted a different way to apply CogDim to constraint diagrams by combining novices' feedback on the interpretation of the notation to analysis dimensions. Starting with the dimensions to test if the activity is acceptable was suggested on the tutorial (Green & Blackwell, 1998). However, when I personally met with Professor Green, he suggested starting with an activity to test the dimensions. As a result two steps were required. The first step was identifying the activities. The second step was to apply each dimension to each activity. This framework was applied to both versions: constraint diagrams and natural language.

The CogDim framework has had several decades of development (Green, 1989; Green & Blackwell, 1998; Green, 2000; Green, 2006; Green, et al., 2006) and it is a widely accepted approach that has credibility in the software evaluation community. To recap, it is the only HCI evaluation approach intended to evaluate languages (Blackwell, et al., 2000), and thus there are many studies that have used CogDim to evaluate the cognitive features of the languages. For example, CogDim was used to cognitively compare Prograph and LabVIEW (Green & Petre, 1996), to evaluate Pursuit (Modugno, et al., 1994), to evaluate PrologSpace (Yazdani & Ford, 1996), to evaluate design rationale representation (Shum, 1991), to evaluate continue-patterns in spreadsheets (Hendry, 1995), to evaluate modification in language such as Basci and Prolog (Roast & Siddiqi, 1996), to evaluate domain-specific languages (Pereira, et al., 2008), to evaluate distributed notations (Green, et al., 2006), and to develop a framework called Representation Design Benchmarks (Yang, et al., 1997), which measured the static representations for a language. Moreover, Microsoft used this framework as a vocabulary for evaluating the usability of their C# (Clarke, 2001), .NET development tools (Clarke, 2004), and an object oriented application programming interface (Clarke & Becker, 2003).

In this evaluation study it is used to examine the relative strengths and weaknesses of Constraint Diagrams notation and conventional notations in terms of the cognitive facilitation or impediments of these different representations. We compared the evaluation of Constraint Diagrams and of Natural Language by running a usability study to determine if users would be able to use constraint diagrams to accomplish a set of tasks. The objectives of this study were to predict the difficulties that may be faced when working with these tasks, such as interpreting or constructing constraint diagrams. Moreover, we aim to provide a rich exploration of advantages and disadvantages of using constraint diagrams for specifying programs compared to natural language, and to point out the key problem areas in constraint diagrams that need to be redesigned. We hope to provide an analysis to determine if users would be able to use constraint diagrams to accomplish a set of tasks.

To evaluate a notation using a cognitive dimensions framework, two steps were required (Green, 2000). The first step is to decide what activities a system is desired to support. An activity is described at a rather abstract level in terms of the structure of information and constraints on the notational environment. Each activity has its own

requirements in terms of cognitive dimensions, and demands a different profile to support them, and is described by the information structure and the constraints on the notational environment. As a result, the second step is to analyse the system and determine how it lays on each dimension. A dimension is a property of the notation or a descriptor which captures an aspect of the nature of the notation that affects the way users may interact with it. The dimensions are a discussion tool, which help to provide designers with a framework to detail its analysis and to focus on its issues. To apply CogDim, every dimension should be described (Blackwell, et al., 2001) with illustrative examples, case studies, and associated advice for designers. In general, an activity such as exploratory design, where software designers make changes at different levels, is the most demanding activity. This means that dimensions such as *viscosity* and *premature-commitment* must be low while *visibility* and *role-expressiveness* must be high (Green, 2000).

In this chapter we evaluate the notation itself, constraint diagrams notation. However, we are going to use representative examples from a Patient Record System which is an application designed to have lots of features yet be easy to use, and it is described in full detail in chapter 3, section 3.6. Before applying individual dimensions to the activities, we will first identify the activities.

4.2. Notational Activities

According to the importance of an activity to CD users, the six activities (Green 2000) are: exploratory design, modification, incrementation, searching, transcription and exploratory understanding, which are now described with examples:

An activity such as exploratory design, which is the most required program specification task, is used for adding new components and changing an existing structure. Green introduced this activity as hacking or in other words as programming on the fly when sketching out designs. For example, the types of tasks for this activity are: (1) adding a new set called gender with only two disjoint subsets called male and female as in Figure 4.5. This set is a subset of an existing set called Patient which has two disjoint subsets called alive and dead as in Figure 4.1. Both male and female subsets intersect with both alive and dead; (2) adding legs and feet of a spider; (3) inserting a new partitioning for a set, in the case of the program specification and the patient record system; (4) adding a new invariant that extends an existing invariant and

is extended by another existing invariant; and (5) an event could be related to a different un-extending or un-extended invariant.

Modification is an activity that changes an existing structure without adding any new information. Green illustrated this activity with a simple example: “*At first, young people put books or music CDs on their shelves in random order; later they impose a bit of a system; still later, they probably revise the system, as their tastes change or as their collection grows*” (Green, 2000). For our study, we will use tasks such as: (1) change a spider with three feet from being located in two intersected sets to a set that includes these two intersected sets; (2) change the target of a relation from being a set, to being a spider; (3) change gender from set with only two elements to denote male and female to two disjoint subsets of patients called male and female and each is intersected with both alive and dead, as in the case of the program specification; (4) regarding the patient record system, an event could be related to a different extended or extending invariant.

Incrementation is an activity that involves adding more information without changing the structure. Green gave an example of the telephone device with a memory where a user can incrementally store the phone numbers in the device memory (Green, 2000). In our case, the user can, for example, add (1) a label to an existing component, (2) a spider, in the case of the program specification, (3) in the patient record system, a patient called Peter (usually by using an event).

Searching is applied when the user is looking for information. Green provided an example of searching the telephone memory for the forgotten and invisible number of Aunt Mary (Green, 2000). For example, the user is searching for: (1) a spider with three feet and two legs located in two intersected sets; (2) the source or target of the relation; (3) all the subsets of a set; (4) all the elements of a set; (5) all the supersets of a set, in the case of the program specification; (6) in the patient record system, a patient called Peter (usually by using a query); (7) the event’s class; (8) the extending classes; and (9) the extended classes. We believe that in the case of modelling systems, searching is not a very demanding activity. However, it is extremely demanding in the case of web pages.

Transcription is coding or copying the specification from one representation to another. It is usually an undemanding activity for software designers who use it as an aid to

evaluate the specification that is written in a specific representation. However, as Green illustrated, it is the main activity for the telephone device with a memory where a user needs to convert written telephone number to a sequence of pressed buttons (Green, 2000). For our study, any CD could be converted to (1) natural language, (2) symbolic language, (3) OCL, (4) Class diagrams, or (5) Z notation.

Exploratory understanding is a higher level activity which is more related to both notational tools and distributed notations, and is not relevant to specifying programs. Furthermore, it is the least well specified activity and Green did not provide examples that allow us to easily use it. We believe that this activity is not related to CD because CD is proposed as a program specification language and not as a language for discovering structure. As a result, this activity is beyond the scope of this research.

4.3. Dimensions

The list of the activities and the dimensions has been changed. In (Blackwell, et al., 2001), Green and colleagues wrote that there are new dimensions created by others such as free rides and creative ambiguity, but in another paper (Blackwell & Green, 2003) he suggested that these two dimensions are already describing the detail of existing dimensions such as closeness of mapping and provisionality, respectively.

The second step is applying each dimension to each activity. In the most recent version of CogDim, there are fourteen dimensions. We will give their definitions in Table 4.1 as described in (Blackwell & Green, 2003) .

These dimensions are now illustrated by Green with the following examples:

The viscosity dimension is about the resistance to change. Green provided examples of this dimension such as manually changing US spelling to UK spelling in a large-sized document. This type of action requires repetition and thus Green called it repetition viscosity. Another example Green provided was inserting a new figure which will require additional actions such as updating all later figures, updating their cross-reference within the text, and also updating the list of figures and the index. He called this type of change entailing further actions a knock-on viscosity. For exploratory design and modification activities, only if the notation's viscosity is low, then this notation meets this dimension. However, for incrementation, searching and transcription

activities which are not about changing components, it is not important whether the notation meets or fails to meet this dimension.

Table 4.1 Cognitive Dimensions Definitions

Cognitive Dimension	Definition
Viscosity	the cost of making changes
Hidden dependencies	the invisible links between components
Abstraction level	(a) combining information together to enhance patterns (b) abstraction barrier
Premature commitment	the constraints on the order of doing things
Visibility	the ability to easily find components
Secondary notation	the extra informal information
Closeness of mapping	(a) closeness of the representation to the domain (b) free rides (inferences)
Consistency	achieved when similar semantics are expressed in similar syntax
Diffuseness	the verbosity of the language
Error-proneness	inviting error
Hard mental operations	achieved when the notation does not provide any aid with mental operations
Progressive evaluation	the ability to check the work at any stage
Provisionality	(a) the degree of commitment to actions (b) creative ambiguity
Role-expressiveness	the readability of the purpose of each component

The hidden dependencies dimension relates to the not-fully-visible relationships between different components. Green explained this dimension by providing the example of the HTML links that are fossils because they are pointing to deleted or moved pages. For exploratory design, modification, searching and transcription activities, if the notation's hidden dependencies are low, then this notation meets this dimension. However, for incrementation activity which is not imposing any kind of changes, it is not important whether the notation meets or fails to meet this dimension.

The abstraction level dimension is about grouping elements to be treated as one entity. An example that Green illustrated for this dimension is sentence styles such as setting

all the sentences of level 1 headings to 24-point bold by creating a style called Heading1. For exploratory design, modification, searching and transcription activities, if the notation's abstraction level is high, then this notation meets this dimension. However, for incrementation activity which is not imposing any kind of changes, it is not important whether the notation meets or fails to meet this dimension.

The premature commitment dimension concerns the enforcement to make a decision before having the necessary information due to the constraints on the order of doing things and the guessing of the spatial place. The amateur signwriter is the example Green provided, where the user would guess the width of the wording while writing the sign. For exploratory design, modification and incrementation activities, if the notation's premature commitment is low, then this notation meets this dimension. However, for transcription and searching activities which are not imposing any kind of changes, it is not important whether the notation meets or fails to meet this dimension.

The visibility dimension is about the ability to easily view components. The indexing facilities of telephone directory design do not provide the name of the subscriber that has a specific telephone number, as Green explained. For exploratory design, modification, incrementation, searching and transcription activities, if the notation's visibility is high, then this notation meets this dimension.

The secondary notation dimension relates to the extra and non-official meaning information such as indentation that is usually used in programming languages, which has no meaning for compilers but makes it easy for users to read this pretty-printing as Green illustrated. Another example provided is the convention of reading and writing telephone numbers by splitting them into a group of digits depending on the region, and also reading from left to right. For exploratory design and modification activities, if the notation's secondary notation is high, then this notation meets this dimension. However, for incrementation, searching and transcription activities which are not imposing any kind of changes, it is not important whether the notation meets or fails to meet this dimension.

The closeness of mapping dimension relates to the closeness of the representation to its domain. LabView is an example of a visual programming language that is, as Green mentioned, closely modelled on an actual circuit diagram. For exploratory design,

modification, incrementation, searching and transcription activities, if the notation's closeness of mapping is high, then this notation meets this dimension.

The consistency dimension relates to expressing similar semantics in similar syntactic forms. The example Green gave is in-car audio sets where the keys, instead of moving through the menus, move up or down. For exploratory design, modification, incrementation, searching and transcription activities, if the notation's consistency is high, then this notation meets this dimension.

The diffuseness dimension relates to the verbosity of language. Green gave an example of a verbose language called COBOL where a command such as MULTIPLY A BY B GIVING C is used. For exploratory design, modification, incrementation, searching and transcription activities, if the notation's diffuseness is low, then this notation meets this dimension.

The error-proneness dimension relates to invitations to mistakes. Green provided the example of FORTRAN language where I and O are used as identifiers, but can be confused with one and zero. For exploratory design, modification, incrementation, searching and transcription activities, if the notation's error-proneness is low, then this notation meets this dimension.

The hard mental operations dimension is about the high demand on cognitive resources. The example Green gave is mazes. Physical mazes may depend on memory-less algorithms, but auditing spreadsheets is hard. For exploratory design, modification, incrementation, searching and transcription activities, if the notation's hard mental operations are low, then this notation meets this dimension.

The progressive evaluation dimension relates to checking the work-to-date at any time. The example here is the spreadsheets which frequently re-compute formulas, as Green illustrated. For exploratory design, modification, incrementation, searching and transcription activities, if the notation's progressive evaluation is high, then this notation meets this dimension.

The provisionality dimension relates to the degree of commitment to actions for serving for the time being. The use of pencils is the example that Green provided for this dimension. This usage allows designers to make fuzzy marks to mean something may go somewhere. For exploratory design, modification, incrementation, searching and

transcription activities, if the notation's provisionality is high, then this notation meets this dimension.

The role-expressiveness dimension is about the readily inferred purpose of a component. A radio circuit diagram can be looked at and its parts can quickly be picked out at different stages by any electrical engineer. For exploratory design, modification, searching and transcription activities, if the notation's role-expressiveness is high, then this notation meets this dimension. However, for incrementation activity which is not imposing any kind of changes, it is not important whether the notation meets or fails to meet this dimension.

In the next section, we will use examples from the Patient Record System case study, which was mentioned earlier in chapter 3, to evaluate the dimensions within each activity.

4.4. Examples from the Patient Record System

The CD notation is used to provide the specification of the Patient Record System case study as described in 3.6. In this section we will choose from that case study some of the diagrams that used CD notation which were used previously in the specification of the Patient Record System. We have picked two examples (Figure 4.1, Figure 4.13 and Figure 4.14) that we believe cover most of the CD notation's components. Figure 4.1 is taken partially from Figure 3.5; and Figures 4.13 and 4.14 are partially taken from Figure 3.20. These examples will be augmented by other components in this section in order to show all possible cases when applying the activities, and to check the cognitive dimensions for these cases. The following figures will show constraint diagrams as odd-numbered figures; each is followed by an even-numbered figure to show natural language statements.

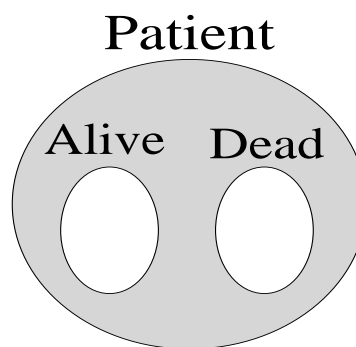


Figure 4.1 CD diagram to represent the Patient set with two disjoint subsets

Figure 4.1 is a diagrammatic example of sets and subsets. It is described in Figure 4.2.

Patient is defined to be a set. This set is partitioned into only two disjoint subsets: those who are Dead and all the others who are Alive. The partition of Patient which is inside that set but outside its two subsets contains no elements, and this represents the empty set. The two subsets are identified by the names Alive and Dead. This expresses that every known patient is either Alive or Dead.

Figure 4.2 NL statement to represent the Patient set with two disjoint subsets

Figure 4.2 is the NL version of the constraint diagram presented in Figure 4.1. As for the previous figure, this is an example of sets and subsets. However, it is represented by a statement.

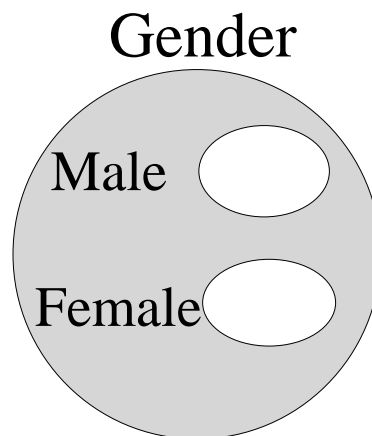


Figure 4.3 CD diagram to represent the Gender set with two disjoint subsets

Figure 4.3 is another diagrammatic example of sets and subsets. It is described in Figure 4.4.

Gender is defined to be a set. This set is partitioned into two disjoint subsets: those who are Male and all the others who are Female. The partition of Gender which is inside that set but outside its two subsets contains no elements, and this represents the empty set. The two subsets are identified by the names Male and Female. This expresses that every known gender is either Male or Female.

Figure 4.4 NL statement to represent that the Gender set with two disjoint subsets

Figure 4.4 is the NL version of the constraint diagram presented in Figure 4.3 to represent an example of sets and subsets.

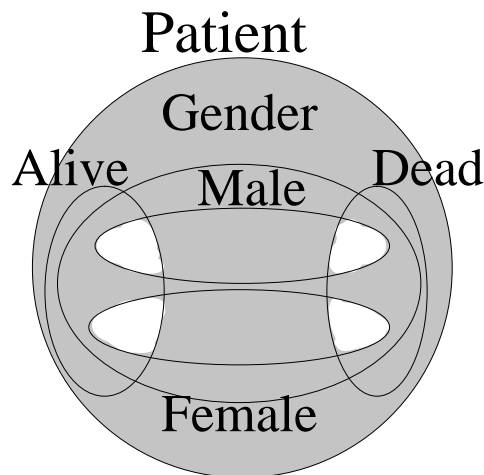


Figure 4.5 CD diagram to represent the Patient set with five subsets

Figure 4.5 is another diagrammatic example of sets and subsets. It is described in Figure 4.6.

Patient is defined to be a set. This set is partitioned into two pairs of two disjoint subsets. The first pair of the two disjointed subsets is about those who are Dead and all the others who are Alive. The other pair is about the two disjointed subsets: Male and Female. These latter subsets are also subsets of the Gender set which is a subset of Patient set. The partition of Patient which is outside the Gender set contains no elements, and this represents the empty set as well as the partition of Gender which is inside that set but outside its two subsets contains no elements. Moreover, the partition of Gender which is not the intersection between the Patient's subsets and Gender's subsets contains no elements. There are only four zones that may contain elements: the intersections of Alive and Male, Alive and Female, Dead and Male, and Dead and Female.

Figure 4.6 NL statement to represent the Patient set with five subsets

Figure 4.6 is the NL version of the constraint diagram presented in Figure 4.5 to represent an example of sets and subsets.

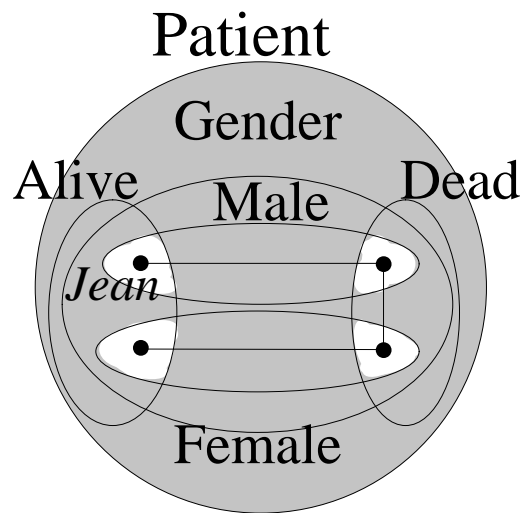


Figure 4.7 CD diagram to present the five subsets of the Patient set along with a spider

Figure 4.7 is another diagrammatic example of sets and subsets. It is described in Figure 4.8.

Figure 4.8 is the NL version of the constraint diagram presented in Figure 4.7 to represent an example of sets and subsets.

Patient is defined to be a set. This set is partitioned into two pairs of two disjoint subsets. The first pair of the two disjointed subsets is about those who are Dead and all the others who are Alive. The other pair is about the two disjointed subsets: Male and Female. These latter subsets are also subsets of the Gender set which is a subset of Patient set. The partition of Patient which is outside the Gender set contains no elements, and this represents the empty set as well as the partition of Gender which is inside that set but outside its two subsets contains no elements. Moreover, the partition of Gender which is not the intersection between the Patient's subsets and Gender's subsets contains no elements. There are only four zones that may contain elements: the intersections of Alive and Male, Alive and Female, Dead and Male, and Dead and Female. Furthermore, there is a patient called Jean.

Figure 4.8 NL statement to present the five subsets of the Patient set along with a spider

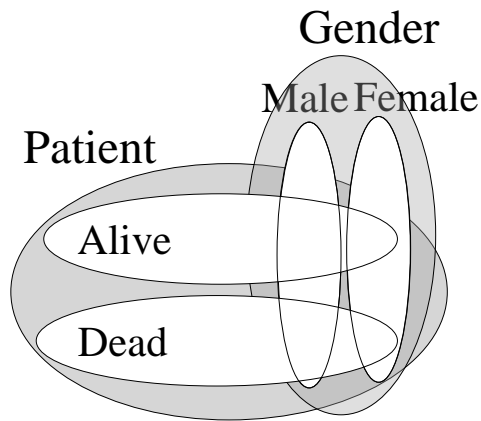


Figure 4.9 CD diagram to present the two sets Patient and Gender along with their subset

There are two sets: patient and gender where each of them has two disjoint subsets alive and dead, and male and female, respectively. The partitions outside the subsets but inside the sets contain no elements.

Figure 4.10 NL statement to present the two sets Patient and Gender along with their subsets

Figure 4.9 is another diagrammatic example of sets and subsets. It is described in Figure 4.10.

Figure 4.10 is the NL version of the constraint diagram presented in Figure 4.9 to represent an example of sets and subsets.

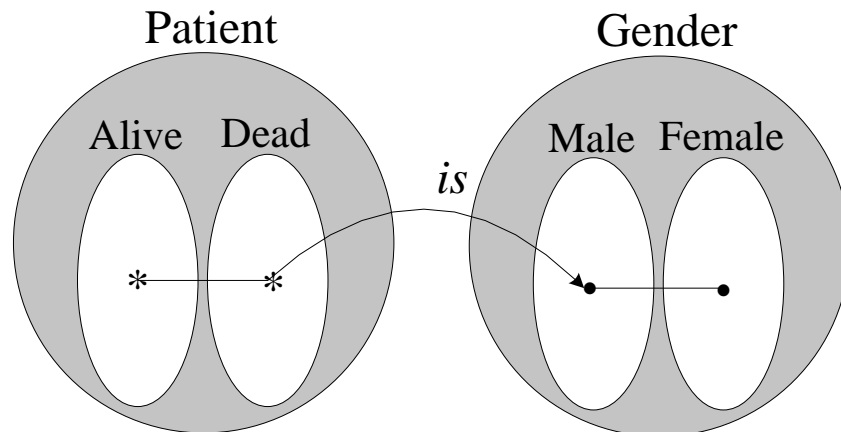


Figure 4.11CD diagram to present that each Patient is either a Male or Female only

Figure 4.11 is another diagrammatic example of sets and subsets. It is described in Figure 4.12.

There are two disjoint sets: Patient and Gender. Each of them has two disjoint subsets. Patient has Alive and Dead whereas Gender has Male and Female. The partitions of Patient and Gender that are inside them but outside their subsets contain no elements. Each patient is either a Male or Female only.

Figure 4.12 NL statement to represent that each Patient is either a Male or Female only

Figure 4.12 is the NL version of the constraint diagram presented in Figure 4.11 to represent an example of sets and subsets.

Figure 4.13 is another diagrammatic example of sets and subsets. It is described in Figure 4.14.

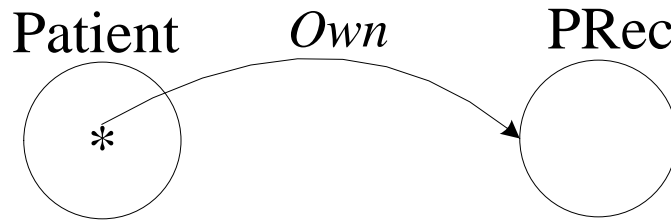


Figure 4.13 CD diagram to present that each Patient has one or many PRec files

Each patient owns one or more PRec file.

Figure 4.14 NL statement to represent that each Patient has one or many PRec files

Figure 4.14 is the NL version of the constraint diagram presented in Figure 4.13 to represent an example of sets and subsets.

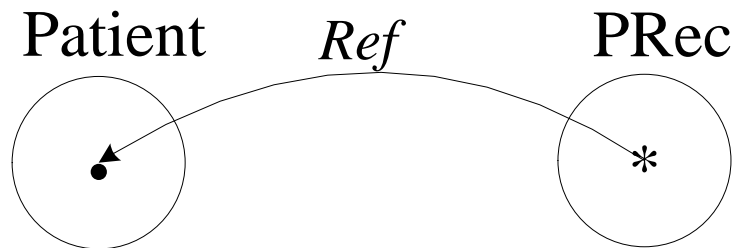


Figure 4.15 A CD diagram to represent that each PRec file refers to only one patient

Figure 4.15 is another diagrammatic example of sets and subsets. It is described in Figure 4.16.

Each PRec file refers to only one patient.

Figure 4.16 NL statement to represent that each PRec file refers to only one patient

Figure 4.16 is the NL version of the constraint diagram presented in Figure 4.15 to represent an example of sets and subsets.

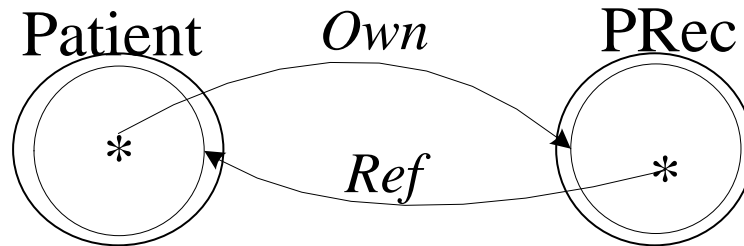


Figure 4.17 CD diagram to present the problem of the differences between domain and habitat

Figure 4.17 is another diagrammatic example of sets and subsets. It is described in Figure 4.18.

For each patient who owned a PRec file, that PRec file refers to all patients.

Figure 4.18 NL statement to present the problem of the differences between domain and habitat

Figure 4.18 is the NL version of the constraint diagram presented in Figure 4.17 to represent an example of sets and subsets.

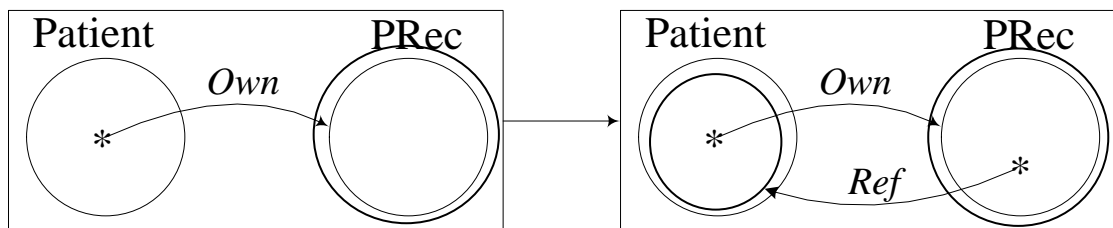


Figure 4.19 A CD diagram to present the generalized version of the CD

Figure 4.19 is another diagrammatic example of sets and subsets. It is described in Figure 4.20.

For each patient who owned a PRec file, that PRec file refers to only that patient.

Figure 4.20 NL statement to represent two relations at the same time

Figure 4.20 is the NL version of the constraint diagram presented in Figure 4.19 to represent an example of sets and subsets.

4.5. Profiles

A profile (Blackwell, et al., 2001) is the integration of the activities with the cognitive dimensions. The profiles together form a discussion portfolio.

4.5.1. Exploratory Design Activity

To recap, this activity relates to adding new components and changing existing structures. The following figures will be used to explain this activity under each dimension.

The viscosity dimension

This is a very important dimension for the exploratory design activity because to lower the cost of this activity, which is about adding and changing components, viscosity must be as low as possible. A set with two disjoint subsets, as in Figure 4.1, could be augmented by having another set called *Gender*, as in Figure 4.5, which itself has two disjoint subsets, *Male* and *Female*, as in Figure 4.3, where each of them intersects with the other two disjointed subsets, *Alive* and *Dead*, as shown in Figure 4.5. Any place outside the intersecting of the two disjointed subsets with the other two disjointed subsets is an empty place as in Figure 4.5.

The sentence that describes Figure 4.5 as shown in Figure 4.6 is harder to read, change and understand than the diagram in Figure 4.5, which means NL can be very viscous and can make the exploratory design activity very difficult. However, using CD as in Figure 4.5 shows why CD is not a viscous language. We changed the non-empty places from two to four easily and we added three new sets easily as well.

Thus, CD notation is a good language to be used for exploratory activity because its viscosity is low. In general, the CD notation meets this dimension for this activity while the NL notation fails to meet it.

The hidden dependencies dimension

In the CD notation, components such as subsets or sources and targets for arrows, are defined clearly and easily, so that a user can visually spot the dependencies. For example we can tell that set *Patient* in Figure 4.5 has many subsets. In general, using a contour in CD makes it clear that it could contain something while a dot will not. Also,

using a line will depend on having things to be joined as one entity, while using an arrow which presents a relation will depend on having a source and a target related to that relation.

However, without careful treatment the order of reading CD can raise an issue of hidden dependencies. For example, Figure 4.13 is the a design of the case where all patients have one or more *PRec* files while Figure 4.15 represents another example of designing a case where each *PRec* file has only one patient. These two cases could be designed as one diagram such as Figure 4.17 to represent that for each patient who owns a *PRec* file, that *PRec* file refers to that *patient*. However, this figure is an example of a not well-matched-to-meaning diagram due to the problem of the difference between domain and habitat (Stapleton & Delaney, 2008). This happened because of the order of reading and drawing the diagram. In this case order of reading could be seen as a hidden dependency.

Reading tree was proposed to reduce the hidden dependencies problem by enforcing the reading order. However, accessing the right information from the reading tree could be difficult in complex cases and require coordination between the tree and the diagram. As a way to solve this, explicit reading order was proposed as a method in (Fish, et al., 2003) and another method of implicit reading order was proposed in (Howse & Schuman, 2005). The first method is useful for reading, but for huge programs this method would increase the required mental operations. Furthermore, as in Figure 4.17, it will cause the problem of the differences between domain and habitat. On the other hand, another method is a generalized version of the CD notation (Stapleton & Delaney, 2008), where a diagram is viewed as a sequence of images which provide the reader with the ability to know exactly how somebody has drawn the diagram, which will provide the exact order of reading the diagram. Thus this last method, as in Figure 4.19, will decrease the mental operations needed to understand the diagram.

In NL, a lot of dependencies are defined in one place and used in another. NL suffers from hidden dependencies which mean that to sketch an existing design you should explore the dependencies to understand whether the augmentation is valid and if so where the location should be. Thus, both CD notation and NL notation fail to meet this dimension.

The abstraction level dimension

The set-theoretical concept and the use of Euler-based diagrams provide the CD notation with a high level of abstraction. This can help a designer sketching the system at high level and avoids too much detail by encapsulating fragments. Despite the fact that abstraction will make the language difficult to learn, it is one of the reasons for the development of OO programming, the solution to viscosity problems, if any, increasing the comprehensibility of a language, and increasing the protection against error-proneness; but it could increase hidden dependences.

The CD notation is based on familiar diagrams such as Euler and Venn and thus the abstraction here is incremental as we can see in the examples of the invariants provided in Chapter 3, Section 3.6, where there was no need to think about variable declarations.

The NL notation, on the other hand, provides users with a detailed explanation because it is a narrative language. Thus it is difficult to apply abstraction to NL, which means, before sketching an existing design, the user should review the existing context word by word and then apply the sketches. Thus, CD notation meets this dimension while NL notation fails to meet it.

The premature commitment dimension

If we want to draw a set called patient which has a subset called alive that contains a spider, then the order of drawing these three components is not important. However if we want to draw an arrow to denote a relation, then a partial order is important; i.e. it is important to have the source and the destination before the relation, but the order of drawing the source or the destination is not important. The same with the spider legs: we cannot draw a leg before having a foot. As a result when dealing with the sets, set relations or elements, there is no order in drawing. The order is only required when dealing with elements relations which are very helpful, to prevent errors and to keep the consistency of the design.

The CD notation sometimes forces you to think ahead and make certain decisions. However, this enforcement is needed to prevent mistakes. For example, Figure 4.1 represents a set called patient with two disjoint subsets, and the place outside these subsets is shaded which means empty. So, to add an element to patient, it will be in alive or dead. Figure 4.7 represents that there is at least one patient who may be a male or female and who may be alive or dead. So the decision we have to make is to add a

spider with four feet and three legs to the diagram to represent this situation and thus this preserve the validity of the diagram. Also, in Figure 4.9, we add a relation between gender and patient sets, but if the requirements were not clear about the set relations between the four subsets, then we would adopt the general Venn theme, which is the intersection to be the relation between them, unless otherwise stated.

If we analogise to the Green and Blackwell example of the signwriter (Green & Blackwell, 1998), we can tell that the CD notation has no such problem because whenever a new subset is added there will be spatial place and if it is too small we can play with the shading notion which is part of the CD system. As a result, premature commitment for guessing the order is high and for guessing the spatial place is low for exploratory design.

However, for NL, the order of sketching is very important. For example, we cannot start saying that alive is related to patient because it is vague. The order here should be to start by defining whether alive and patient are sets or spiders, then later by defining the relation – whether it is a set relation or an element relation, and so on. Indeed in NL, the size of the writing is important because measuring the font size determines the available space. Thus, both CD and NL notations fail to meet this dimension.

The visibility dimension

This dimension is important for solving problems. The CD notation could increase visibility because elements are associated directly with its sets and sets associated directly with its subsets and supersets as well as arrows which are associated with its sources and targets. Also, it is easy to know the order to read CDs because they have the implicit reading tree option which simplifies the diagrams: the explicit reading tree option used with a complex problem, or the generalized CD option which provides a visual reading order of sequences.

In the odd-numbered Figures from 4.1 to 4.17, the places that may contain elements and the empty places are all clearly visible. Moreover, the relations between the sets, subsets and elements are all obvious and visible.

However, using the NL notation to find the information that is needed but is hiding in documentation, the user has to search many pages to find such information. For example, using sentences as used in the even-numbered Figures from 4.2 to 4.18 for

designing shows the invisibility of a valid description of a patient without using some mental operations. Thus, CD notation meets this dimension while NL notation fails to meet it.

The secondary notation dimension

There is no secondary notation for sets and elements. The CD notation does not enforce the convention of reading from left to right. Instead, the reading order starts with the source of an arrow. Furthermore, the CD notation has the option of choosing how to read the diagram. For example, the combined diagrams from both Figure 4.11 and Figure 4.13 is an example of an implicit reading tree representing the same information that is in both Figure 4.17, which represents the generalized CD, and Figure 4.18 which is presented in NL notation.

In general, the CD notation cannot prevent colouring the diagrams, adding textual comments, adding extra reading trees or choosing the location to place things. In this activity, secondary notation is very helpful to explain complex diagrams. However, the high use of secondary notation could increase the complexity of the diagram and the need for hard mental operations. Due to the diagrammatic grouping nature of the CD notation which already gives more information, there is no need to have extra information.

On the other hand, NL notation which allows a substantial number of comments (Green & Petre, 1996) has the benefit of secondary notation. Many comments can be used to explain something. In general, secondary notation in both CD and NL languages can be used to amend the low expressiveness deficiencies which would prevent the representations from being isomorphic of what they represent and thus affect their closeness of mapping. Thus, both CD and NL notations meet this dimension.

The closeness of mapping dimension

The CD notation has a closeness of mapping property because it provides descriptions of objects and behaviours that are related to the represented world. For example, an arrow is well matched to its meaning because it provides information that the source is related to the target.

The CD is a very intuitive notation in being well-matched to meaning (Stapleton & Delaney, 2008). Moreover, Euler diagrams have free rides (Shimojima, 2004) which

have been empirically evaluated (Shimajima & Katagiri, 2008). Thus CD which is an Euler-based notation, has free rides which are inferences that are explicit within a diagram, such as the transition property of the subset relation which leads to free rides. On the other hand, these inferences would have to be derived when using the NL notation. For example, Figure 4.5 shows that there are only four valid places for a spider to be, which is difficult to show by the NL notation as in Figure 4.6 the NL notation suffers from a lack of closeness of mapping due to its textual nature. Thus, CD notation meets this dimension while NL notation fails to meet it.

The consistency dimension

CD has formal syntax and semantics and their syntax is simple due to the fact that it is based on simple diagrams. There are a limited number of symbols to represent syntax. Each symbol has only one meaning and there are no two symbols that represent a similar thing. We have not noticed any particular problems of consistency. So, to sketch a design, you are constrained with the syntax and the semantics to help you focus on the design and prevent you from focusing on other details such as choosing a good syntax to present what you sketch.

On the other hand, NL notation sometimes suffers from a lack of consistency. There are many new words, old words, or words imported from different languages, and also a word can have different interpretations depending on the context. Moreover, NL has a thesaurus and synonyms for each single word. For example, in CD notation, a drawn circle means a set, but in NL notation, a word set means different meanings, such as a group, or the status of changing from liquid to solid which has a totally different meaning. As a result, in NL notation, it depends on the context, not only on the syntax. Thus, CD notation meets this dimension while NL notation fails to meet it.

The diffuseness dimension

CD compacts information because it uses a small space to represent a lot of information and its syntax is very limited – for example in Figure 4.1 there are three components along with their labels. However, NL uses a lot of symbols or a lot of space to represent the same information. NL is a long-winded language due to its narrative nature. To convert Figure 5-1 into NL we will need many words to say that there is a set called patient which contains two disjoint subsets: alive and dead. Any place outside alive and dead but inside patient is shaded to denote an empty place. It can be seen that we

needed 30 words, two punctuation marks and, with this font size, two lines. Thus, CD notation meets this dimension while NL notation fails to meet it.

The error-proneness dimension

For designing activities, visually grouping related information will help designers to detect patterns. By using the CD notation, it is not easy to make mistakes because diagrammatic errors can easily be spotted; e.g. a relation cannot be drawn without a source and a target and also a spider can't be drawn on the edge of a set. However, errors related to textual labels are not easy to spot. On the other hand, NL errors are very difficult to spot. Thus, CD notation meets this dimension while NL notation fails to meet it.

The hard mental operations dimension

Due to the fact that CD has free rides, this reduces the number of hard mental operations. Figure 4.5 shows that the intersection between the area of gender outside male and female and the area of alive gives a shaded area to indicate an empty area; this is a free ride. However, since NL does not have the free rides advantage and some information needs to be clearly stated or be derived, this means there is a need for mental operations. For example Figure 4.2 states that "The partition of Patient which is inside that set but outside its two subsets contains no elements, and this represents the empty set", which means that the designer needs to distinguish between inside and outside and to find out which of them has elements and which does not. This could be more difficult if the number of insides and outsides increases as in Figure 4.6, which the author herself finds it difficult to state in NL and easier in CD. Thus, CD notation meets this dimension while NL notation fails to meet it.

The progressive evaluation dimension

This dimension can be successfully applied if there is a digital environment involved, such as the example of recomputed formulas in spreadsheets that was mentioned earlier. However, assuming a manual environment, we can stop in the middle of creating a diagram to review the work at any time, at any stage, to check the progress, because as shown in Chapter 3, section 3.6.2, the CD design emphasizes scalability. The same procedure is used for NL to spot the stage of the work. Thus, both CD and NL notations meet this dimension.

The provisionality dimension

By considering Figure 4.8 as an example, it shows that the problem domain has two sets, patient and gender, where each of them has two disjointed subsets, alive and dead and male and female, respectively. This sentence does not tell us anything about the relation between patient and gender, patient and both male and female, or gender and both alive and dead. Figure 4.9 is the CD representation for this sentence. Since we do not know the relation between the sets, CD follows the general Venn theme that all sets intersect unless otherwise stated. This allows creative ambiguity and allows playing around with an idea because we are not sure which way to proceed and the given information does not help us to be too precise about the exact result we were trying to get. This sentence is also a clear example of provisionality in NL. Both CD and NL have provisionality and they make exploratory design easier. Another example is Figure 4.7 which uses legs (lines) between spiders to indicate ambiguity of the place of that spider. This figure represents the valid optional places of an element, but the exact place is not certain. Thus, both CD and NL notations meet this dimension.

The role-expressiveness dimension

This dimension is important for the exploratory design because if the purpose of a component is readily inferred, then the design can be read easily. In CD, each component is distinguished by different graphical devices; e.g. existential spiders have dots which are different from universal spiders that have asterisks and different from sets which have contours, etc. The CD notation makes it easier to sketch a design because it has a high role-expressiveness property. Figure 4.7 shows sets by using contours, spiders by dots and a leg between two dots by a line to denote the relationship between them.

On the other hand, using NL sometimes makes it harder to read the design because the role-expressiveness is not clear. It is not always clear if an individual name is for a spider or a set. This means NL makes it harder to sketch a design because its role-expressiveness is low. Furthermore, as mentioned previously, a word set has different roles such as a group or a process of converting liquid to solid. As a result, the role of a set is an ambiguous word when used by NL notation. Thus, the CD notation meets this dimension while the NL notation fails to meet it.

The results of CD/NL can reduce exploratory design costs as follows:

Table 4.2 Summary of Exploratory Design Activity Profile

Cognitive Dimensions	CD	NL
Viscosity	√	×
Hidden dependencies	×	×
Abstraction level	√	×
Premature commitment	×	×
Visibility	√	×
Secondary notation	√	√
Closeness of mapping	√	×
Consistency	√	×
Diffuseness	√	×
Error-proneness	√	×
Hard mental operations	√	×
Progressive evaluation	√	√
Provisionality	√	√
Role-expressiveness	√	×

According to this table (Table 4.2) CD can reduce exploratory design costs by 12:14 while with NL it is by 3:14.

4.5.2. Modification Activity

Modification is an activity that changes an existing structure without adding any new information.

The viscosity dimension

This is a very important dimension for modification activity because to lower the cost of this activity, which is about adding and changing components, viscosity must be as low as possible. Inheritance structure leads to being viscous, and changing the inheritance means changing the pattern of class. In CD, changing a relation from being a set relation to an element relation is visually easier to do rather in other non-diagrammatic notations. For example, Figure 4.5 can be converted easily into Figure 4.11. However, to change this from Figure 4.6 to figure 4.12, a lot of changes are required. In general, in CD, when changing a spider to a set or vice versa, and also when changing a target of a

relation from set to a spider or vice versa, the change can be done easily and there will be no need for a lot of work to change, and thus CD is a less viscous system.

On the other hand, with NL notation, a lot of work is required to change any type. To change a spider to a set, extra work is needed. This will mean that set rules should be applied as well as grammar and a review will be needed as well; this NL is much harder to modify and has higher viscosity, which is harmful for modification. Thus, CD notation meets this dimension while NL notation fails to meet it.

The hidden dependencies dimension

To recap, in the CD notation, dependencies such as subsets or sources and targets for arrows are defined clearly and easily so that a user knows the dependencies. For example, we can tell that set Male in Figure 4.5 has no subsets and the relation's source and target in Figure 4.11 has no hidden dependencies. Thus CD is not a hidden dependencies notation.

However, in NL, a lot of dependencies are defined in one place and used in another, as in figures 4.6 and 4.12. When changing something, a review is needed many times, and thus NL is much harder to modify and it has higher hidden dependencies. Thus, CD notation meets this dimension while NL notation fails to meet it.

The abstraction level dimension

As mentioned before, the set-theoretical concept and the use of Euler-based diagrams provide CD with a high level of abstraction. Changing from Figure 4.5 to Figure 4.11 did not affect the abstraction level and also the abstraction level did not prevent any change.

The NL notation, on the other hand, provides users with a detailed explanation because it is a narrative language, and thus it is difficult to apply abstraction to NL, which means that to change something the user should review the context word by word, modify, review spelling and grammar rules, and review the context, which means it is much harder to modify due to its lower abstract level. Thus, CD notation meets this dimension while NL notation fails to meet it.

The premature commitment dimension

Modifying Figures 4.5 and 4.11 has no requirements for any premature commitment. The modification was straightforward. However, it is not the case with the NL notation,

Figures 4.6 and Figure 4.12. Thus, CD notation meets this dimension while NL notation fails to meet it.

The visibility dimension

This plays an important role in modifying activity to ensure consistency. The modification of Figure 4.5 to 4.11 is easy because all the components are visible and there is no hidden information that needs to be derived which is not the case for NL, Figure 4.6 and Figure 4.12. Despite the fact that modification by using NL requires review of the whole of the document, it may cause inconsistency. The CD notation is much easier than the NL notation to use to modify specifications because it has higher visibility. Thus, CD notation meets this dimension while NL notation fails to meet it.

The secondary notation dimension

Despite the fact that there is no secondary notation for sets and elements, CD cannot prevent any extra information that is not part of the actual design being added by colouring the diagrams or using textual comments. In this activity secondary notation can be used fruitfully to understand a complex diagram or to explain the reasons behind a change. When changing Figure 4.5 to Figure 4.11, we can add a comment explaining the reasons behind changing the relation type from set relation to element relation. Thus, both CD and NL notations meet this dimension.

The closeness of mapping dimension

When changing Figure 4.5 by using the CD notation, it will always maintain the closeness of mapping; for example, a relation will always have a source and a target as in Figure 4.11. On the other hand, NL does not have closeness of mapping due to its textual nature – see Figures 4.6 and 4.12. Thus, CD notation meets this dimension while NL notation fails to meet it.

The consistency dimension

With simple syntax, changing the structure is low cost. Whereas the CD notation has simple syntax with formal semantics, NL notation does not. So to change from Figure 4.5 to Figure 4.11, it will be necessary to change the set relation between Patient and Gender to be an element relation between these sets. So it is consistent in CD notation because a set relation has only one meaning which includes containment, intersection or disjoint, and element relation is presented by an arrow. However by using NL, it is

difficult to know the type of relation unless stated or derived from the context. Thus, CD notation meets this dimension while NL notation fails to meet it.

The diffuseness dimension

Unlike NL notation, CD notation has a limited number of components and much information can be derived as free rides, which implies that a modification task is easy to do. Figures 4.5 and 4.11 used a small place to represent, while Figures 4.6 and 4.12 needed to be diffused to state that a patient is only alive or dead and each alive or dead patient is either male or female only. Thus, CD notation meets this dimension while NL notation fails to meet it.

The error-proneness dimension

It is easy to spot any mistakes when the specification is changed using CD notation. For example, when modifying Figure 4.13 or Figure 4.15 to become Figure 4.17, we spotted that there were differences between domain and habitat, and thus we knew that there would be an issue with the reading. Thus, in this case, we either use a reading tree or the generalized CD notation. There are other visually spotted mistakes such as placing a spider on the edge of a set or deleting the source of an arrow or placing a set as a source of an arrow. However, errors related to labels are not easy to spot. As a result, the CD notation is much easier to modify because it has less error-proneness. In NL, mistakes can easily be made due to the nature of textual languages. NL has high error-proneness which means it is much harder to modify. Thus, CD notation meets this dimension while NL notation fails to meet it.

The hard mental operations dimension

Modifying from Figure 4.5 to Figure 4.11 is straightforward and there is no need to recall any information. The CD notation can reduce the need for memory or mental calculations because it shows constraint and displays information in visual ways. It is much easier to modify by using CD because fewer hard mental operations are required.

By using the NL notation, we can describe Figure 4.11 by saying that there are two disjointed sets which have a spider with three feet and two legs. It is hard to modify this sentence because it needs hard mental operations. For example, to replace this spider by a set, we cannot replace word by word; i.e. it is incorrect to say there are two intersected sets which have a set with three feet and two legs. Mental operations are needed to understand that a spider is an element which means it should belong to a set, and has

feet and legs, and to remember that feet should be in different places; no two feet are in the same place. Also the replacing set may be a superset of the intersected sets, a subset of them, or a disjoint set. It is much harder to modify by using NL because hard mental operations are required. Thus, CD notation meets this dimension while NL notation fails to meet it.

The progressive evaluation dimension

We can stop in the middle of modifying a diagram to review the work at any time, at any stage, to check the progress. The same procedure is applied to NL to spot the stage of the work. The type of the used environment, manual or digital, will lead to a manual or digital progressive evaluation, respectively. Thus, both CD and NL notation meet this dimension.

The provisionality dimension

For example, to change a set relation as in Figure 5-5 to an element relation without knowing a specific patient's exact place, as in Figure 5-11, allows creative ambiguity because we are not sure which way to proceed and the given information does not help us to be too precise about the exact result we are trying to get. So, in Figure 4.11, we can say there is a patient Jean and because we do not know if she alive or not and if she is male or not we will represent Jean same way as in Figure 4.11 but we will use an existential spider rather than the universal spider. This sentence is also a clear example of provisionality in NL. Both CD and NL have provisionality and they make modification, even with missing information, easier. Thus, both CD and NL notations meet this dimension.

The role-expressiveness dimension

Since in CD each component is distinguished by different graphical devices, the purpose of each of them is readily inferred. To modify Figure 4.5 to Figure 4.11, we look for a contour and modify a set relation to be an element relation; we add an arrow. The CD notation makes it easier to modify because it has high role-expressiveness. Figure 4.15 shows a set by using a contour, and the relation between a universal spider and an existential spider by an arrow. On the other hand, using NL sometimes makes it harder to modify the design because the role-expressiveness is not clear. It is not always clear if an individual's name is for a spider or a set. This means NL makes it harder to modify a design because its role-expressiveness is low, and thus the modification will

depend on the context. Thus, CD notation meets this dimension while NL notation fails to meet it.

The results of CD/NL reduction of modification costs are:

Table 4.3 Summary of Modification Activity Profile

Cognitive Dimensions	CD	NL
Viscosity	√	×
Hidden dependencies	√	×
Abstraction level	√	×
Premature commitment	√	×
Visibility	√	×
Secondary notation	√	√
Closeness of mapping	√	×
Consistency	√	×
Diffuseness	√	×
Error-proneness	√	×
Hard mental operations	√	×
Progressive evaluation	√	√
Provisionality	√	√
Role-expressiveness	√	×

According to this table (Table 4.3) CD can reduce modification costs while the NL cost is 3:14.

4.5.3. Incrementation Activity

Incrementation is an activity that involves adding more information without changing the structure.

The viscosity dimension

Since this activity is not about changes to components, this dimension is not important to evaluate. As a result it is not important if CD or NL are viscous or not, which means that both languages are fine for this activity under this dimension and both do not increase the incrementation cost. Thus, both CD and NL notation meet this dimension.

The hidden dependencies dimension

Inserting a new label into the design doesn't require an extreme number of individual actions nor a change at the plan level. However, there should be a graphical device to label. Also, this label should not have been used before. Unlike NL, every dependency is explicit and clearly visible. We can add a spider in any contour in Figure 4.5 and even if we do not know its exact place we will not face any difficulties in inserting that spider as shown in Figure 4.7 because by using spider's legs and feet we can show all the possible places, which indicates the lack of knowledge of the exact place. Nevertheless, there could be hidden dependencies when adding a new arrow in Figure 4.13 or Figure 4.15 if the order of the reading was not clear. Figure 4.17 which represents this issue has the problem of the difference between domain and habitat. This issue could be solved by using reading trees or the generalized CD notation.

In general this dimension does not affect this activity much because incrementation is about adding, not modifying. As a result both languages are fine for incrementation regarding hidden dependencies. Thus, both CD and NL notation meet this dimension.

The abstraction level dimension

Abstraction is useful either to lower the viscosity or to stimulate the user's conceptual structure. Since incrementation is not about changes, viscosity is not an issue for this activity and also it is not about capturing the structure as shown in Figures 4.6 and 4.7. Thus, since this dimension will not cause any problems to this activity, both the CD and NL languages do not increase incrementation costs. Thus, both CD and NL notation meet this dimension.

The premature commitment dimension

Since new data is being created, this dimension is vital to this activity. In Figure 4.5, to insert a set along with its subsets that contain spiders as in Figure 4.7, the order of the insertion of these components is not important. In this case whenever a piece of information comes we can insert it easily. However, to insert a new label such as Male, there should be a graphical device to label and this label should be unique. We cannot make decisions before we have the information we need about which graphical device – which is a subset in this case – to attach the label to. Moreover, to insert an arrow which represents elements' relations, its source and target should be inserted first. Otherwise, we cannot insert this arrow. In general, when inserting sets, set relations or elements,

there is no order to follow. The order is only required when dealing with elements' relations, which is very helpful to prevent errors and to keep the consistency of the design. The CD notation sometimes forces you to think ahead and to make certain decisions to prevent mistakes. For example, the decision we are forced to make when adding a label is to have an existing graphical device first, and thus this preserves the validation of the diagram. Another example is inserting a subset into a set which has many subsets and there is no spatial place to add more subsets. In CD, if the spatial place is too small we can play with the shading notion which is part of the CD system.

On the other hand, due to the nature of NL, the order of insertion is very important. For example we cannot say that alive is related to patient because it is vague. The order here should be to start by defining whether alive and patients are sets or spiders, then later by defining the relation as a set relation or element relation, and so on. Indeed in NL the size of the writing is important because measuring the font size determines the available space. Thus, both CD and NL notations fail to meet this dimension.

The visibility dimension

For this activity which is about inserting components, visibility is useful for error-checking only. For example, in Figure 4.5, to insert a new existential spider in Patient, it is visible that there are only four zones where this patient could be placed and it would be an error to place it in other zones as shown in Figure 4.7. As a result both languages do not increase the incrementation costs. Thus, both CD and NL notation meet this dimension.

The secondary notation dimension

Inserting new information using CD or NL: this dimension will not have an effect on this activity. We could insert new components in Figure 4.5 with a different colour. For example, if inserting spiders into Patient as in Figure 4.7, we could for example draw the spider feet on the Female set by a pink colour and on the Male set by a blue colour. However, since the use of colours is not part of the notation, it is the same whether colours are used or not. As a result, both languages will not increase the incrementation cost under this dimension. Thus, both CD and NL notation meet this dimension.

The closeness of mapping dimension

In the represented world, two things are related by a specific relation. In the CD notation, a uniquely identified arrow, as in Figure 4.11, is used to relate a source to its

target. Thus, inserting a relation is very closely related to what we want to describe because a source and a target to this relation will be either existing or inserted along with the relation. Unlike NL, CD is close to mapping. Thus, CD notation meets this dimension while NL notation fails to meet it.

The consistency dimension

Since CD has simple syntax, incrementation is low cost. Meaning is clear in CD; for example, a dot means a spider, and a relation head points to the target while its end points to the source as in Figure 4.11. However, NL has a wider syntax and the same word may mean different things depending on the context. To insert a spider using NL we would use a lot of symbols or a lot of space for defining the spider, its type, describing the location of that spider, and explaining its relations to the existing components. Unlike NL, CD has a limited number of components and much information can be derived as free rides, which implies that an incrementation task is easy to do. Thus, CD notation meets this dimension while NL notation fails to meet it.

The diffuseness dimension

The CD notation compacts information because it uses a small space to represent a lot of information and its syntax is very limited. In Figure 4.5, to insert a spider in Patient as in Figure 4.7, we would add a dot in a specific spatial place. However, to insert a spider using NL we can use a lot of symbols for representing a spider such as element, member, individual ...etc. Thus, CD notation meets this dimension while NL notation fails to meet it.

The error-proneness dimension

It is easy to spot that there is a mistake, for example in Figure 4.5, inserting a spider as in Figure 4.7 but on the edge of a set, or inserting an arrow as in Figure 4.11 but without a source – these are visually spotted errors. However, errors related to labels are not easy to spot. As a result, CD is much easier to use for incrementation because it has less error-proneness. In NL, mistakes can easily be made due to the nature of the textual languages. NL has high error-proneness, which means it is much harder to spot insertion errors. Thus, CD notation meets this dimension while NL notation fails to meet it.

The hard mental operations dimension

Incrementation is straightforward and there is no need to remember any information. The CD notation can reduce the need for memory or mental calculation because it

shows constraint and displays information in visual ways. It is much easier to insert when using the CD components because then less difficult mental operations are required. For example, to insert a spider in Figure 4.5 to be as in Figure 4.7, there will be nothing to remember and no mental operations are needed. However, to insert a spider using NL, mental operations are needed to understand the context and remember what and where the grouped related information is, and to insert the new information into that group, which indicates that it is much harder to increment by using NL. Thus, CD notation meets this dimension while NL notation fails to meet it.

The progressive evaluation dimension

Both languages support stopping in the middle of incrementation to review the work at any time, at any stage, to check progress or to ascertain the stage of the work. As previously stated, this review could be done manually or digitally depending on the environment used. Thus, both CD and NL notation meet this dimension.

The provisionality dimension

In Figure 4.5, adding a spider to denote that there is a patient, doesn't require a lot of work, but the location of that spider should be known. In case the information on the spider's location is missing, CD allows us to present this by having a spider foot and legs as in Figure 4.7. Even the NL notation allows creative ambiguity by not being precise. Both CD and NL have provisionality and they make insertion, even with missing information, easier. However, CD insertion will represent missing information by giving all the possible cases, which will keep the diagram valid. On the other hand, NL will be ambiguous and will give many different inferences. Thus, both CD and NL notation meet this dimension.

The role-expressiveness dimension

As previously discussed, since in CD each component is distinguished by different graphical devices, the purpose of each of them is readily inferred. To insert a set, we will definitely add a contour and to insert a relation, we will add an arrow. For example in Figure 4.5, to add a specific patient we will use an existential spider to denote that patient and if we do not know the exact place of that patient we will use feet and legs to denote the possible places of that patient as in Figure 4.7. the CD notation makes it easier to insert new information into the design because it has high role-expressiveness. On the other hand, NL has lower role-expressiveness. However, to insert the text new

information, role-expressiveness is not a critical dimension. As a result, both languages can reduce the incrementation costs under the role-expressiveness dimension. Thus, both CD and NL notation meet this dimension.

The results of CD/NL reduction in incrementation costs are:

Table 4.4 Summary of Incrementation Activity Profile

Cognitive Dimensions	CD	NL
Viscosity	√	√
Hidden dependencies	√	√
Abstraction level	√	√
Premature commitment	×	×
Visibility	√	√
Secondary notation	√	√
Closeness of mapping	√	×
Consistency	√	×
Diffuseness	√	×
Error-proneness	√	×
Hard mental operations	√	×
Progressive evaluation	√	√
Provisionality	√	√
Role-expressiveness	√	√

According to this table (Table 4.4) CD can reduce incrementation costs by 13:14 while NL would be by 8:14.

4.5.4. Searching Activity

Searching is applied when the user is looking for information.

The viscosity dimension

Since it is not an important dimension for searching activity because this activity is not about changes to components, it is not essential if the CD or NL notations are viscous or not. So both of them are fine for this activity under this dimension and neither of them increases the searching cost. Thus, both CD and NL notations meet this dimension.

The hidden dependencies dimension

The important dimension to look at for searching is the hidden dependencies. In Figure 4.11, searching for a target of a relation in the design will simply require us to look for the graphical device near to the arrow head. Unlike NL, every dependency is explicit and visible. So, the NL notation can slow up searching. As a result, CD can reduce the search costs by speeding up the search. Thus, CD notation meets this dimension while NL notation fails to meet it.

The abstraction level dimension

Abstraction is about grouping and visually relating information together. Unlike NL, CD is a scalable notation which allows hierarchical searching by using overviews to locate areas for a more detailed search. For example, in Figure 3.20 we can search for more information of the *HProf* set and we can see from this figure that more information is located on *HCS* class and thus we can search there. Thus, CD notation meets this dimension while NL notation fails to meet it.

The premature commitment dimension

Searching has nothing to do with making decisions prior to having the information that we need. So premature commitment is not vital for searching activity using either CD or NL notations. Thus, both CD and NL notation meet this dimension.

The visibility dimension

In the CD notation, the visibility is good because, as in Figure 4.7, elements are associated directly with their sets and a set is associated directly with its subsets and supersets, as well as, in Figure 4.11, an arrow which is associated with its sources and targets. Also, it is easy to know the order in which to read CDs because they have an implicit reading tree which simplifies the diagrams, or an explicit reading tree which is used with complex problems. In NL to find the information needed in a document, the user has to search many pages to find similar information. Thus, CD notation meets this dimension while NL notation fails to meet it.

The secondary notation dimension

To search for information using CD or NL: this dimension will not have an effect on this activity. Perhaps using secondary notation here will speed up the search by allocating the location of the searched-for component. As a result, both languages will

not increase the search cost under this dimension. Thus, both CD and NL notations meet this dimension.

The closeness of mapping dimension

An effective mapping can speed up the interpretation and can lead to fewer errors. Since CD enables inference operations and is close to mapping as previously discussed, it reduces the search costs; which is not the case with NL. Thus, CD notation meets this dimension while NL notation fails to meet it.

The consistency dimension

In CD, searching for a subset as in Figure 4.1 means searching for a set contained in another set, which is always true and could be visually spotted easily. However, searching for a subset using NL is not easy, especially if the word subset was not explicitly mentioned and has to be derived from the context. Thus, CD notation meets this dimension while NL notation fails to meet it.

The diffuseness dimension

Due to CD's diagrammatic nature, it compresses information into a small space because it has a limited number of components and much information can be derived as free rides, which reduces the search costs. For example, Figure 3.20 represents the whole system of Patient Record System using simple components in a small space, and locates the place of the additional information from the extended classes. However, NL is a diffused language with many synonyms, antonyms and a thesaurus. Thus, CD notation meets this dimension while NL notation fails to meet it.

The error-proneness dimension

Searching for both valid and invalid results is easy with CD notation. The user will easily spot that there is a mistake if searching for a spider on the edge of a set or searching for a source of an arrow which is not there. As a result, CD is much easier to use for searching for valid results because it has less error-proneness. However, in NL searching for a set as a source of a relation is difficult to spot as an invalid search. Thus, CD notation meets this dimension while NL notation fails to meet it.

The hard mental operations dimension

CD reduces hard mental operations because diagrams are spatially grouped by related information such as the sets in Figure 4.5, which provide rapid access, reducing the

need to match symbolic labels, acting as a memory resource to aid users, detecting patterns, enabling inference operations such as in Figure 4.5, which indicates that the shaded area is an empty area, and compacting information into a small space, as in Figure 4.11. However, NL requires hard mental operations to understand the context and remember what and where the grouped related information is, which indicates that it is much harder to search using NL especially if the search includes negatives and self-embedding. Thus, CD notation meets this dimension while NL notation fails to meet it.

The progressive evaluation dimension

A user is unlikely to stop in the middle of searching to spot the stage of the work. Therefore, this dimension is irrelevant to this activity. However, CD and NL do not prevent you from doing that. Thus, both CD and NL notations meet this dimension.

The provisionality dimension

This dimension is about creating ambiguity and non-precise information. By searching using CD, it is easy to distinguish between information and missing information. For example, in Figure 4.7, by searching for a Jean, we conclude that Jean is a patient but other information is missing. It is not known whether Jean is male or female, nor whether she is alive or dead. However, distinguishing is difficult in the case of NL. Thus, CD notation meets this dimension while NL notation fails to meet it.

The role-expressiveness dimension

This dimension is important for reducing the cost of searching because when searching for a set in CD, we are looking to find a contour, while if we are searching for a relation, we will look for an arrow. For example, in Figure 4.7, we are searching for a specific patient called Jean and thus we are looking for a spider and because we have found this spider with feet and legs we know that the location of that spider was not available during the insertion time. The CD notation makes searching easier because it has high role-expressiveness whereas NL sometimes makes it harder to read the design because the role-expressiveness is not clear. Thus, CD notation meets this dimension while NL notation fails to meet it.

The results of CD/NL reduction of search costs are:

Table 4.5 Summary of Searching Activity Profile

Cognitive Dimensions	CD	NL
Viscosity	√	√
Hidden dependencies	√	×
Abstraction level	√	×
Premature commitment	√	√
Visibility	√	×
Secondary notation	√	√
Closeness of mapping	√	×
Consistency	√	×
Diffuseness	√	×
Error-proneness	√	×
Hard mental operations	√	×
Progressive evaluation	√	√
Provisionality	√	×
Role-expressiveness	√	×

According to this table (Table 4.5) CD supports searching by reducing the searching costs while the NL cost is 4:14.

4.5.5. Transcription Activity

Transcription is coding or copying the specification from one representation to another.

The viscosity dimension

This is not an important dimension for transcription activity because this activity is not about changing components. Transcription does not impose any kind of changes, as shown in transcribing from Figure 4.1 to Figure 4.2. So this dimension does not affect this activity much – since there is no change – and thus both languages are fine for transcription regarding viscosity, and neither increases the cost of transcription. Thus, both CD and NL notations meet this dimension.

The hidden dependencies dimension

Transcribing a target of a relation, such as in Figure 4.11 to Figure 4.12, will simply require knowing its relation and its source which can all be seen by the graphical

devices: the arrowhead points to the target and its end points to the source. Unlike NL, every dependency is explicit and easily visible. As a result, CD can reduce the transcription costs. However, NL may cause transcription difficulties because when transcribing without knowing the hidden dependencies, this will cause invalid cases such as transcribing Figure 17. Unlike the CD notation, the NL notation increases transcription costs. Thus, CD notation meets this dimension while NL notation fails to meet it.

The abstraction level dimension

In general, abstraction is useful for capturing the structure for ease of transcription activity and the CD notation visually groups related information. For example, Figure 4.7 has higher abstraction than Figure 4.8, and therefore the CD notation reduces transcription cost which is not the case with NL. Thus, CD notation meets this dimension while NL notation fails to meet it.

The premature commitment dimension

Since no new information is being created in the structure by the transcription activity, no decisions are required to be made before having any necessary information, as shown in transcribing from Figure 4.7 to Figure 4.8. As a result, premature commitment is not vital for transcription activity using either CD or NL notations, and thus both notations meet this dimension.

The visibility dimension

For this activity, visibility is required to ensure consistency, which is useful for error-checking only. To transcribe from Figure 4.7 to Figure 4.8 it is visible that the exact place of the spider Jean is not known. So the known and unknown information are visible using the CD notation. However, since this dimension is not vital for this activity, both languages do not increase the transcription costs. Thus, both CD and NL notations meet this dimension.

The secondary notation dimension

In general, having extra information could help during the transcription activity, but it is not essential. The CD notation cannot prevent the colouring of the diagrams, adding textual comments or choosing the size of the components. However, due to the diagrammatic grouping nature, CD does not need to have extra information – as in Figure 4.7, where we did not need to add a comment to say that we did not have any

information about the exact place of the spider Jean. In fact, both languages do not increase transcription activity costs for this dimension. Thus, both CD and NL notations meet this dimension.

The closeness of mapping dimension

Using the CD notation speeds up the interpretation of the design that is needed to make the transcription easier, and this could lead to fewer errors than with NL. The CD components are generally well-matched to meaning – for example, the arrows in Figure 4.11, which says that any patient is either a male or female. Thus, CD notation meets this dimension while NL notation fails to meet it.

The consistency dimension

In CD, transcription is a low cost because a set is visually spotted as only a contour without any mental cost; also a relation is spotted as an arrow. However, transcription using NL is not easy, especially if the word patient did not explicitly refer to a set, which gives an option of referring to a spider, and this will need a deriving operation since NL lacks free rides. Also the word ‘set’ itself, as we previously mentioned, has several interpretations, such as a group, or a process of converting liquid to solid. As a result, the meaning of the word set in NL is ambiguous. Thus, CD notation meets this dimension while NL notation fails to meet it.

The diffuseness dimension

With transcription activity, it is easy to use the CD notation because it has a limited number of components, which reduces transcription costs such as transcribing Figure 4.11. However, it is difficult when using NL because it is a diffused language, with many synonyms and antonyms, especially if negatives and self-embedding are included. Thus, CD notation meets this dimension while NL notation fails to meet it.

The error-proneness dimension

It is easy to spot that there is a mistake such as transcribing a spider on the edge of a set or transcribing an arrow without a source or a target. As a result, CD is much easier to use for transcription because it has less error-proneness. However, in NL transcribing a set as a source of relation is difficult to spot as an invalid search, especially with negatives and self-embedding. Thus, CD notation meets this dimension while NL notation fails to meet it.

The hard mental operations dimension

Transcription is straightforward and there is no need to remember any information. CD can reduce the need for memory or mental calculation because it shows constraints and displays information in visual ways. It is much easier to transcribe when using CD because fewer hard mental operations are required. For example, transcribing Figure 4.11 to Figure 4.12 is easy, but vice versa is not. However, to transcribe a set using NL, mental operations are needed to understand the context and remember its subsets, members, supersets, and relations with other sets and non-members, which indicates that it is much harder to transcribe using NL, especially if negatives and self-embedding are included. For example Figure 4.2 states that “The partition of Patient which is inside that set but outside its two subsets contains no elements, and this represents the empty set”; this needs some mental operations to imagine the inside and outside of a set and to determine which of them has the elements and which does not. Thus, CD notation meets this dimension while NL notation fails to meet it.

The progressive evaluation dimension

We can stop in the middle of transcription to review the work at any time, at any stage, to check the progress or to ascertain the stage of the work in both CD and NL languages. This review could be manual or digital depending on the environment used. Thus, both CD and NL notations meet this dimension.

The provisionality dimension

When transcribing using the CD notation, it is easy to distinguish between information and missing information, such as in Figure 4.7. However, this distinction is difficult in the case of NL. Thus CD notation meets this dimension while NL notation fails to meet it.

The role-expressiveness dimension

This dimension is important for reducing the cost of transcription because if the purpose of a component is readily inferred then the transcription can easily be done. The CD notation makes it easier to transcribe a design because it has high role-expressiveness, as shown in Figure 4.7, where a spider is shown by a dot and a set by a contour; whereas the NL notation sometimes makes it harder to read the design because the role-expressiveness is not clear. Thus, CD notation meets this dimension while NL notation fails to meet it.

The results of CD/NL reduction in transcription costs are:

Table 4.6 Summary of Transcription Activity Profile

Cognitive Dimensions	CD	NL
Viscosity	√	√
Hidden dependencies	√	×
Abstraction level	√	×
Premature commitment	√	√
Visibility	√	√
Secondary notation	√	√
Closeness of mapping	√	×
Consistency	√	×
Diffuseness	√	×
Error-proneness	√	×
Hard mental operations	√	×
Progressive evaluation	√	√
Provisionality	√	×
Role-expressiveness	√	×

According to this table (Table 4.6) CD can reduce transcription costs while the NL cost is 5:14.

The summary of applying each dimension to each activity for CD language is shown in Table 4.7 to answer whether the notation is increasing the activity costs or not:

Table 4.7 CD Integrated Profile

Cognitive Dimensions	Exploratory Design Activity	Modification Activity	Incrementation Activity	Searching Activity	Transcription Activity
Viscosity	√	√	√	√	√
Hidden dependencies	×	√	√	√	√
Abstraction level	√	√	√	√	√
Premature commitment	×	√	×	√	√
Visibility	√	√	√	√	√
Secondary notation	√	√	√	√	√
Closeness of mapping	√	√	√	√	√
Consistency	√	√	√	√	√
Diffuseness	√	√	√	√	√
Error-proneness	√	√	√	√	√
Hard mental operations	√	√	√	√	√
Progressive evaluation	√	√	√	√	√
Provisionality	√	√	√	√	√
Role-expressiveness	√	√	√	√	√

Moreover, the summary of applying each dimension to each activity for NL language is shown in Table 4.8:

Table 4.8 NL Integrated Profile

Cognitive Dimensions	Exploratory Design Activity	Modification Activity	Incrementation Activity	Searching Activity	Transcription Activity
Viscosity	×	×	√	√	√
Hidden dependencies	×	×	√	×	×
Abstraction level	×	×	√	×	×
Premature commitment	×	×	×	√	√
Visibility	×	×	√	×	√
Secondary notation	√	√	√	√	√
Closeness of mapping	×	×	×	×	×
Consistency	×	×	×	×	×
Diffuseness	×	×	×	×	×
Error-proneness	×	×	×	×	×
Hard mental operations	×	×	×	×	×
Progressive evaluation	√	√	√	√	√
Provisionality	√	√	√	×	×
Role-expressiveness	×	×	√	×	×

4.6. Discussion

Overall, it seems that the CD profile is promising; in general, the cost of the activities using CD is less than the cost using NL. Only in one dimension, premature commitment, did CD not satisfy two activities. However, this dissatisfaction is needed to ensure the diagram's formal validity. We are glad we have not faced the case where high viscosity and high premature commitment are combined because it would be the worst problem, as Green discussed (Green & Blackwell, 1998). Exploratory design and incrementation activities have a high level of premature commitment, which is a problem. However, since viscosity is very low this lets premature commitment be less costly, since bad guesses can easily be corrected.

CD notation is cognitively better than NL notation according to our single case study. It is meant to specify different situations and it may be used for other complex case studies that have different processes and may give different results. We covered all of the five activities that could be used for program specification and for each of these activities we looked at all fourteen of the different dimensions.

Despite the fact that Green (Green, 2000) advised evaluating the notation and its environment, and recommended a paper-based environment, CD is well used by editors. Since there is no specific editor for CD, despite the basic editor (Gil & Sorkin, 2013), most users use Microsoft Visio to draw diagrams (Halpin, et al., 2003)

The trade-offs between the different dimensions were obvious. For example, in NL to solve the lack of visibility, a secondary notation is required. Another thing to be noted is the trade-off within the dimension itself. For example, secondary notation in CD could provide bracketing information, prevent ambiguities, help inexperienced users to understand the complex diagrams, help experienced users to express more complex specifications, and provide unique semantics. However, it will not keep the basic syntax of the notation, makes the syntax of a diagram more complicated, and reduces the simplicity of a notation.

In this study, we used dimensions to examine the activities and we used activities for finding the undesired degree of a dimension. For example, we faced a problem of high level premature commitment with CD in exploratory design and incrementation activities and luckily the level of viscosity was low, which means correction guesses were cost-free. Moreover, if the levels of viscosity and premature commitment were

high, we could adopt a different notation or a different medium by making a draft version of the system in a lower viscosity medium and then transferring it to the target medium, which breaks this process into two activities: exploratory design activity followed by transcription activity.

Cognitive dimensions of notations helped us in understanding the nature of the structure of CD notation. Despite the fact that CD is a complex notation, this framework is suitable for exploring the cognitive aspects for this complex notation. This study provides a full profile for each activity that can be carried out when using this notation, supported by a number of cognitive aspects to understand how this notation can be used in specifying software.

Although we found that CD notation is cognitively better than NL in terms of supporting program specification, this evaluation focused on only one case study, and different case studies may be more complex than the one we investigated. For this evaluation we picked a selected range of examples which we thought would enrich the evaluation.

The choice of CD and NL notations might impact on the evaluation because if another notation was chosen which was close to CD, such as any other formal notation, we might find that the other notation was cognitively better than CD. Furthermore, the cognitive differences between the diagrammatic and sentential notations that were discussed in detail in (Larkin & Simon, 1987) might affect the comparison as well. Perhaps using these two notations to represent the patient record system also affected the system itself and vice versa.

Chapter 5 Experiment 1: Interpretation of Constraint Diagrams

This experiment attempts to evaluate the potential utility of CD for users who have relatively little experience in specifying programs by comparing two different groups, to check how easy it is for such users to learn about CD and interpret specification expressions in comparison with using NL expressions. Section 5.1 introduces the experiment. Section 5.2 represents the experimental design. Section 5.3 explains the pilot experiments. Section 5.4 shows the results of the experiment. Finally, section 5.6 is the discussion.

5.1. Introduction

This chapter describes a comprehensive investigation to explore the usability of CD as a program specification language. This experiment focused on two cognitive activities for learning: thinking and reasoning. It took the form of a web-based competition in which 53 participants were given instructions and training either on CD or equivalent NL specification expressions, and then responded to multiple-choice questions requiring the interpretation of expressions in their particular notation. It was predicted that participants using CD notation would take longer over the training, need more time to answer the questions, be less confident about their answers, and obtain lower correct interpretation scores, because they had no prior experience of CD notation.

This empirical study is aimed at evaluating how difficult CD notation is to interpret by investigating whether it can support people with relatively little technical background, novice users of software systems specification, and whether it is possible for users to understand constraint diagrams easily and rapidly. It is also aimed at checking the effectiveness of CD notation by finding out whether learning is going to be a major task compared with other notations. We performed this experiment to identify the benefits of using CD in specifying programs.

This study considered the interpretation of specifications, rather than their construction, because it is the first step in evaluating this notation and we wanted to test how easily and rapidly this notation could be learnt and interpreted. Thus, this

experiment's target was novice users with little experience in specification, and without any experience of using CD in specification. As a result of having novice users as a target and in order to answer the question of whether CD can work for complex problems, we examined their interpretation with basic concepts and simple questions to find out if there was a fundamental difference between CD and NL, which has not been done before.

5.2. Experimental Design

We will now consider: (a) the design of the experiment, (b) the training approach to familiarize people with the constraint diagrams, (c) the measurements of the results as evidence, (d) the predictions, (e) the design of the examples and questions, (f) the concepts to include in the experiment, and (g) the appropriate number of questions.

(a) A between-participants design was adopted with two separate groups and two different representations: constraint diagrams and natural language. Two different versions of the same material, questions and examples were created, which were informationally equivalent (Larkin & Simon, 1987). Participants were randomly assigned to the CD group or the NL group. Although some people would argue that a within-participants design would have made comparison easier, our experiment was to compare their understanding of the concepts in a specific representation without any prior experiments that bias familiarity.

(b) In spite of the fact that there are individual differences in cognitive styles, training is the best solution to persuade users to learn about constraint diagrams because differences disappear with training (Frandsen & Holder, 1969; Blackwell, 1997). Training users to understand constraint diagrams allows a comparison between these two different representations, despite the fact that NL notation is familiar to users while CD is not.

(c) In order to evaluate how effective the different notations were, we measured how accurately and how quickly they were used. The measures used were: time spent on the training examples, time taken to answer the questions, percentage of correct answers, level of confidence rating, and number of returns to the examples. The time was recorded between the example or question online page being loaded and the *Next* button being pressed.

(d) It is predicted that NL will be better because it is familiar and there is no need to learn a new notation before working on the tasks. Given the complexity of CD, the experimental hypothesis states that participants given the CD notation will obtain fewer correct answers in a longer time with a lower level of confidence rating and many returns to examples. This experiment has the version of the specification (CD or NL) as the independent variable; whereas correct answers, confidence, question time, example time, and returning to examples are the dependent variables.

(e) The domain used for training examples was different from the one used for questions. Each example that introduced a concept provided training from a domain called “Video Rental Service” while the questions’ domain was called “Patient Record System”. There were two design options: (1) an example followed by three questions or (2) all examples followed by all questions. The benefits of (1) are to look at each concept separately and to examine their understanding of that concept individually, which gives the opportunity for incrementally constructing knowledge. This will help us to identify how difficult a concept was. We also believe that learning should be active and meaningful so learners can apply what they have learned, and control the learning process. On the other hand, (2) would be quite a heavy load for learning in one go and needs high mental operations. We adopted the first option so that they could learn each concept by elaborating each concept.

(f) There were eight concepts covered in this experiment: Sets and Types, Members of sets, Set relationships, Relationships, Spiders, Spider relationships, Invariant and An Event Specification. Every concept was introduced using training examples that were followed by three questions.

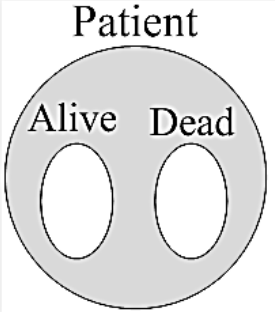
(g) A pilot experiment, which was a face-to-face experiment between the participants and the experimenter, was conducted in order to obtain the level of difficulty and the number of questions. As a result, it was determined that 24 questions would be appropriate.

5.3. Pilot Experiments

Pilots are used as a method to evaluate the material and the software. Pilot 1 was aimed at evaluating the material testing the constraint diagrams notation. The purpose of this pilot was to gather valuable data that would increase the value of this material. It was run as a one-to-one tutorial which helped in understanding the value of the gathered

data. During pilot 1, four participants, who were undergraduate informatics students from the University of Brighton who had finished a course on constraint diagrams, were monitored and were asked some questions.

Question 8:




From this diagram, can a *Patient* be a child?

☐ Yes
☐ No
☐ Not Specified

How confident you are about your answer?

☐ Not at all Confident
 ☐ Not Very Confident
 ☐ Not Confident
 ☐ Neutral
 ☐ Confident
 ☐ Very Confident
 ☐ Extremely Confident

 Example


 Next

Figure 5.1 A snapshot from Pilot 1

The developed java application had examples of a constraint diagram's concepts, each followed by questions to examine participants' understanding. They were asked to rate their confidence about each answer. Pilot 1 had five examples and fifteen questions (each example followed by three questions). Figure 5.1 is a snapshot of question number 8 from this pilot. The participants answered by choosing one of three options: *Yes*, *No* or *Not Specified* as shown in Figure 5.1. If they were satisfied with the information provided, then they would answer the question with *Yes*. However, if they were not satisfied with the provided information, then they would answer the question with *No*. Otherwise, if they felt that some information influencing their decisions was missing, then they would choose *Not Specified*. *Not Specified* means there is insufficient information provided in the question to answer it without any assumptions, which is the case in Figure 5.1 where the provided diagram did not specify anything about being child or adult, and thus the answer here is *Not Specified*. While participants were trying to answer, they could return to the example by pressing a button called *Example* as shown in Figure 5.1. However, if there was no need to return to the example, they could

proceed to the next question by pressing the other button which was called *Next*, except that in the last question (question 15) the *Next* would be called *Finish*. After answering, participants would rate their level of confidence about their answer. A seven-point scale (1= lowest score, and 7= highest score) (Churchill, 1979) is used to measure their confidence: *Not at all Confident*, *Not Very Confident*, *Not Confident*, *Neutral*, *Confident*, *Very Confident*, *Extremely Confident*. However, participants only used five levels. This application was a tutorial with examples and questions; the examples were used to teach participants new notation concepts and the questions to examine how easy the notation explanations were. Participants used a tool which automatically recorded their feedback and answers in a log file to prepare them to be analysed. The time they spent on reading each example and on answering each question was recorded. Also, the number of times they returned to an example was counted. These factors were very helpful in ascertaining which examples were difficult to understand and which questions were difficult to solve.

A final revision of the experiment's material was successfully carried out, and there were many lessons learnt from pilot 1, such as: 20 to 30 questions were a suitable number of questions for the experiment. The number of confidence levels had to be smaller than the one used in the pilot. We reduced the number of levels of confidence from seven to five levels, and from being personal to being specifically about the question rather than the self-esteem. Although it was mentioned in the instructions, the participants did not use the *Example* button, which was a help tool, until it was mentioned personally to them. Some of them were not interested in seeing the example again. Also, the *Next* button was used sometimes without providing an answer. Participants did not understand the option *Not Specified* until its use was explained verbally. The last two points led to the need for a training example to instruct them on using the *Example* button and the *Not Specified* option. Also, the material was revised to only concentrate on the notation itself without the framework.

By adopting the material of pilot 1, pilot 2 aimed at including the lessons learnt from pilot 1 and testing the software itself. The main plan here was to provide an easily accessible online experiment which would avoid installing the software on each PC. We used ASP.Net web application and we tried it with 91 participants who were undergraduate informatics students from the University of Brighton and the University of Qatar. Lots of developments within many months were made in order to make the

software error-free. These tests helped us to monitor the participants' interactions with the software itself, so some software designs were changed. The *Next* button property was changed from invisible to disabled until participants answered. Although in the instructions we mentioned that they would not be able to proceed until they answered, it seemed they forgot this point because when they did not want to answer and they aimed to proceed, they were pressing the only visible button, the *Example* button, which took them back to the example and they believed that the software had crashed.

From these pilot experiments, we now explore the main experiment.

5.4. Experiment

5.4.1. Method

In this experiment, there were two experimental learning trial conditions: (1) The CD group which used the constraint diagrams version of the system for program specifications; (2) The NL group which used a system of program specifications written in natural language. It was a training web-based competition which randomly assigned participants to one version. This learning-based experiment was a tutorial which taught the participants program specification concepts for a CD or an NL representation. The criteria for judging the best performance in the competition is a combination of spending less time learning the new concepts and getting the most answers correct in the least time.

Before starting the competition there was a tutorial on how to use the software. This tutorial was one simple algebra example, as shown in Appendix A, followed by three questions which were randomized between participants. The example used for the tutorial contained a box containing an algebraic statement "The product of 2 and 4 subtracted from x equals 10" and this box was followed by a description "This statement represents the value of x . $x - 2 * 4 = 10$ means $x - 8 = 10$ which means $x = 10 + 8$, So $x = 18$ ". This example and the first question had step-by-step instructions to teach participants about using the training. The second and third questions aimed to test their ability to use it. After finishing the software training tutorial, the competition began with training examples. Each training example was followed by three related questions.

I will explain what the participants could see. Each example had one or more statements (NL version) or an image (CD version) and a description of it followed by a definition.

After studying the example to understand the concept, participants had to press the *Next* button to proceed.

For any question, one or more statements or an image appeared, depending on the representation version (NL/CD). This was followed by a question about it. Participants had the opportunity to return to the related example associated with that question any time before answering by pressing a button called *Example*. They couldn't proceed until they had provided an answer, relying only on information given in the current question, without any assumption from previous questions. Participants answered a question by choosing one of three answering options: *Yes*, *No* or *Not Specified*. For each concept's questions, there might be any possible combination of answers. Consequently, there would not necessarily be one specific answer for all three questions. However, we did not tell the participants that for each training example the answer would consist of one *Yes*, one *No* and one *Not Specified*. There were five levels to measure how hard the question was: *Very Difficult*, *Difficult*, *Intermediate*, *Easy* and *Very Easy*. Participants rated each question on its difficulty, which would show their confidence level in answering. Subsequently, feedback according to the answer was provided. Then participants had to press the *Next* button to proceed.

The measures used were: time spent on the training examples, time spent on the questions, percentage of correct answers and level of confidence rating. Each example had only one button called *Next*. When participants pressed it, the time was recorded starting from the time the example page was loaded until the time of pressing the *Next* button. However, each question had two buttons: the *Example* button and the *Next* button. Participants could either answer the question or press the *Example* button. In contrast, they would not be able to press the *Next* button unless they rated the level of difficulty of answering the question, which couldn't be done without answering the question first. The question time was recorded from the time a question page was loaded until the time participants pressed a button. If participants returned to an example, then the number of returns was recorded as well as the time spent on that example. As a result, the example time would be the total time spent on that example, and the question time would be the total time spent on that question before returning to the example page and after returning to the question page to answer. When participants answered, the number of correct answers for each question was also recorded. Moreover, feedback on that answer was provided after participants chose their level of confidence. Immediate

feedback had a valuable effect on participants' learning. This rating showed how confident at answering they were. It was important to understand whether participants answered depending on what they believed was the correct answer or whether they only wanted to go to the next page in order to proceed. This evidence measured if constraint diagram notation could become familiar to users. They also measured the ease and the rapidity of learning it. Designing experimental software avoids pencil-and-paper experiments. Despite the fact that developing software is a time-consuming method, it is a helpful tool to accurately record the time, and the number of tries at answering. We developed this web-based experiment by using Visual Basic.NET in Microsoft Visual Studio to create an ASP.NET web application. Therefore participants easily accessed the online competition and we successfully avoided installing the application on each workstation.

5.4.2. Subjects

The subjects were undergraduate and postgraduate informatics/computing students at Sussex and Brighton Universities. Since the purpose of this study is to examine the CD notation, not the program specification itself, informatics students were chosen as participants because they can be considered as representative of the target users of CD and they have no background in CD, but they have little knowledge of program specifications. As a result, we targeted participants who were aware of the idea of software design and implementation; therefore they had a little knowledge of program specification without any experience of real-life program specification. They voluntarily participated in the experiment through a competition with six prizes worth in total 100 pounds. They were divided randomly into two learning groups. There were 53 participants (27 NL, 26 CD) who participated online, of which 33 (20 NL, 13 CD) produced usable data because they completed more than half the questions. There was a smaller proportion of female participants, but the ratio of male to female was about the same in the two groups. They were asked to provide their personal information and to both consent to be involved in the competition and read the experiment instructions.

5.4.3. Materials

To recap, there were two versions: CD and NL versions. The inferences which were made in one representation can be made in the other representation as well. Both had the same tutorial, eight examples and 24 questions, but a different representation. They were given a training example and three questions, both with step-by-step instructions,

to train them on how to use the online experiment. After that, the main testing began. Participants studied the concepts provided by the examples. After each example there were three questions that participants had to answer in order to proceed. There were two domains: one for the examples and the other for the questions.

Figure 5.2 and Figure 5.3 are screen-snapshots from the experiment. They were used in the CD version and NL version, respectively. Each represents example 7 in a different representation. Each example has a title of the introduced concept, either a diagram in the case of the CD version or a statement in case of the NL version, followed by a description, definition of important parts in the concept, and the *Next* button. This applied to all examples in the experiment.

Example 7: Invariant

This diagram shows that a Video Copy class (*VC*) maintains a set of *Title* (s) (uniquely identified by elements from given type **T**). Each known *Title* has their own associated *Desc* (Description) (of type **D**), and is either in *InColl* (In-Collection) and in *ExColl* (Ex-Collection), but not in both.

Definition: The general form is:

Core Concept name

Semi-box. *STATE-INVARIANT*

Statements of the core concept go here. They are the conditions that are always true about the class.

This framework represents a core concept (class). Its state-invariant is written as any mix of declarations and predicates (separated by a ‘;’ when they appear on the same line). State-invariants of the core concept are the conditions that are always true about the class.

Next

Figure 5.2 A snapshot of training example 7 using CD

Example 7: Invariant

A *Video Copy* class maintains a set called *Title* (uniquely identified by elements from given type **T**). Each known *Title* has its own associated *Description* (of type **D**), and is either in *In-Collection* or in *Ex-Collection*, but not in both.

Definition:

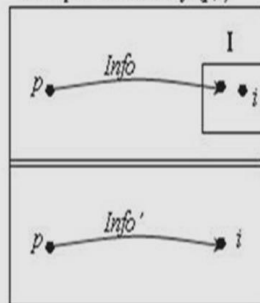
A core concept (class) has two components; a class name and a state-invariant which is written as any mix of statements of declarations and predicates. A State-invariant of the core concept is a combination of conditions that are always true about the class.

Next

Figure 5.3 A snapshot of training example 7 using NL

Question 23

$FP!UpdatePatientInfo(p,i)$



From the given diagram, can a piece of *Info* (Information) be updated?

- ☒ Yes
☐ No
☐ Not Specified

Well Done! The answer is Correct!

How hard did you find the question?

- ☐ Very Difficult
 ☐ Difficult
 ☒ Intermediate
 ☐ Easy
 ☐ Very Easy

Example

Next

Figure 5.4 A snapshot of question 23 using CD

Question 23

An event, called '*UpdatePatientInformation* (p,i)', can be used to update current *Information* for a *patient*, p , so it has some value i afterwards. Before performing this event p should be an element of *Patient* and i should be a value of type **I** which differs from the current value of *Information*; so After performing this event that value will indeed be changed, whilst still preserving the invariant of *PP*.

From the given statement, can a piece of *Information* be updated?

☐ Yes
☐ No
☒ Not Specified

The answer is Wrong! The correct answer is Yes.

How hard did you find the question?

☐ Very Difficult
 ☐ Difficult
 ☒ Intermediate
 ☐ Easy
 ☐ Very Easy

Example
Next

Figure 5.5 A snapshot of question 23 using NL

Figure 5.4 and Figure 5.5 are screen snapshots from the experiment. They are used in the CD version and the NL version. Each represents question 23 in a different representation. A question has a title consisting of its number, either a diagram in the case of the CD version or a statement in case of the NL version, followed by a related question; three answering options: *Yes*, *No* and *Not Specified*; a difficulty level question followed by five levels: *Very Difficult*, *Difficult*, *Intermediate*, *Easy* and *Very Easy*; feedback on the answer; and the *Example* and *Next* buttons. This applied to all questions in the experiment.

A full range of all the examples and questions used in this experiment is in Appendix A. Now we will present the results.

5.5. Results

Overall, it was predicted that the participants in the CD group would find learning concepts and answering questions in the domain harder. It was anticipated that the CD group would have more incorrect answers to the questions, spend longer on giving answers, and be less confident. In this section, we will explore the dropout rate, to

examine which group was happier to stick with the experiment. Then we will investigate the results according to the measurements that we pointed out in the previous sections. We will investigate the performance according to the correct answers, the question time, the confidence rating, the example time, and the number of the returns to the examples. Moreover, we will consider the relations between these measurements: question time and correct time, confidence rating and correct answers, question time and confidence rating, example time and correct answers, example time and question time, and example time and confidence rating.

Further, item analyses using *t* tests or *Chi-squared* tests are needed to find any differential effects of representation for certain questions. It is an approach to give a richer picture of the use of the representations. However, when using repeated tests the chance of finding a false positive is increased. A false positive occurs when finding a significance difference but in reality there is no such significance. Thus, Bonferroni correction is used to lessen the chances of finding a false positive.

The Bonferroni test is used, as needed for this experiment, to adjust the significance levels using the number of comparisons because several tests are done on the same data. By dividing the conventional significance level which is 0.05 by the number of tests which is 24, the adjusted significance level is ($p < 0.002$), which applies to all the *t* tests. It is important to achieve the adjusted significance level for the *t* tests to be counted as significant.

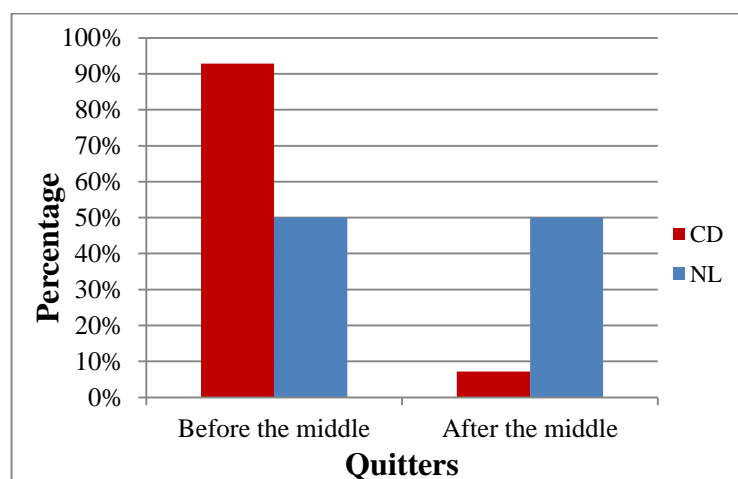


Figure 5.6 The drop-out rate

Some participants withdrew from the experiments over time. In total, 28 of the 53 participants dropped out. The drop-out rate for both groups was the same: 14 for each. As shown in Figure 6.6, before the middle of the experiment 13 participants (93%) out of the 14 dropped out from the CD group, and there were 7 participants (7%) who dropped out of the 14 from the NL group. However, after the middle of the experiment, there was only one participant (7%) who dropped out from the CD group and 7 participants (50%) who dropped out of the 14 from the NL group. Since 53.85% from the CD group and 51.85% from the NL group dropped out from the experiment across time, the overall rate for both groups was approximately the same. To be more precise, there was a pretty constant percentage of drop-out participants from the NL group in both halves of the experiment. However, there was a significant difference for the CD group in the two halves. The CD users who did not withdraw earlier were happier to stick with the experiment. Perhaps their decision to quit was affected by the version of the notation that they received: CD or NL notations. There is a significant difference between the two groups in both halves according to Chi-squared tests (one-tailed), $X^2(1, N = 33) = 0.012$ at the conventional $p < 0.05$, but it is not significant at the Bonferroni adjusted significance level ($p < 0.002$). Moreover, for each half, and also for the overall withdraw rate, there is no significant difference between the two groups. Since participants withdrew at different stages of the experiment, we have chosen to analyse data related to those who completed at least half of the questions.

5.5.1. Correct Answers

This measurement is used to find out which notation would provide more correct answers. This could reflect their understanding of interpretation using that notation. If using CD notation results in having fewer correct answers than using NL notation, then NL notation is better, which is our hypothesis here since CD is a new notation and participants need to learn and understand how this new notation is used and to think and interpret using it.

Figure 5.7 shows the mean number of correct answers for each question for the two groups. The overall mean number of correct answers by group is largely similar across the questions. There appears to be a general trend for the proportion of correct answers to decrease from the earlier to the later questions, although the question-by-question variability increases substantially in the second half of the questions. The mean (and

SD) of the CD and NL groups over all of the questions are 0.729 (0.445) and 0.745 (0.436) respectively.

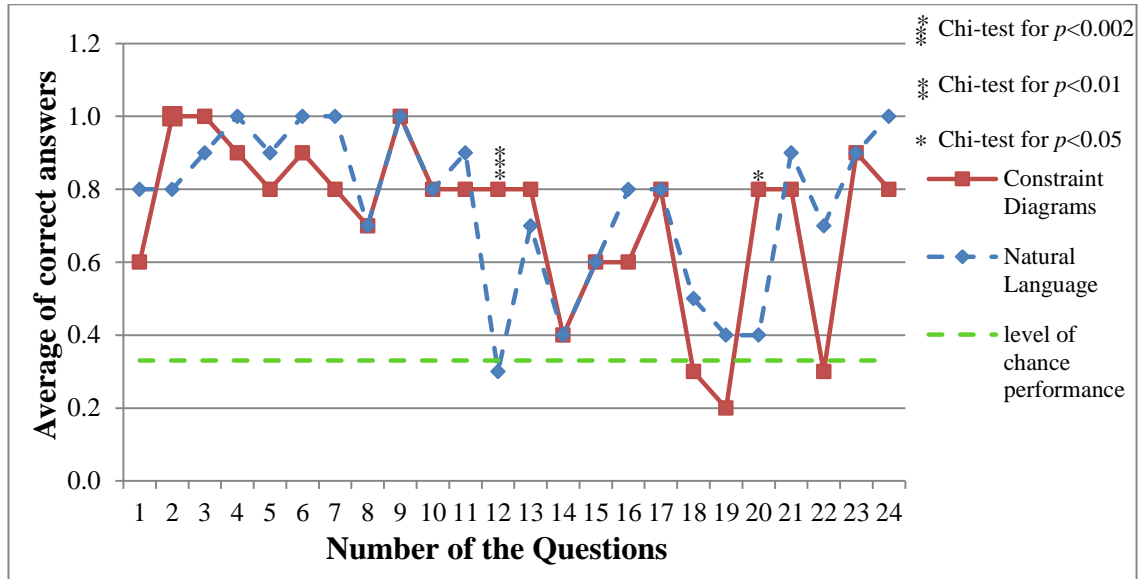


Figure 5.7 Graph of the average of correct answers for the two groups across the 24 questions

By using a mixed design ANOVA (Figure 5.8), there is a significant main within-subjects effect of the two halves of the questions: $F(1,30)=48.76$ for $p<0.01$ and $\eta^2=0.619$. However, there is no significant main between-subjects effect of the two representations: $F(1,30)=0.037$ for $p=0.85$ and the interaction between the two representations for both halves shows that it is not significant; $F(1,30)=1.048$ for $p=0.31$. The graph (Figure 5.7) shows that the second half (each half is 12 questions) is worse than the first for both representations, which indicates that the CD notation is not worse than NL notation. Although the CD group have more correct answers in the first half compared with the NL group, with a mean of 0.846 compared to 0.806, they are worse in the second half with a mean of 0.602 compared to 0.626. However, these mean values are not significant and thus, the lack of significant results indicates that there are no differences between both representations and no within-subjects differences for both halves.

By examining each question individually, we find that there is no significant difference between the two groups in all but two of the questions, according to Chi-squared tests. For question Q12 the difference between the groups is significant, $X^2(1, N = 33) = 0.001$ ($p<0.002$) and similarly for Q20, $X^2(1, N = 33) = 0.034$ ($p<0.05$). The CD group achieved a proportion of correct answers that was greater than or equal to that of the NL groups

in 12 questions. Overall, the CD group did not perform substantially worse than the NL group.

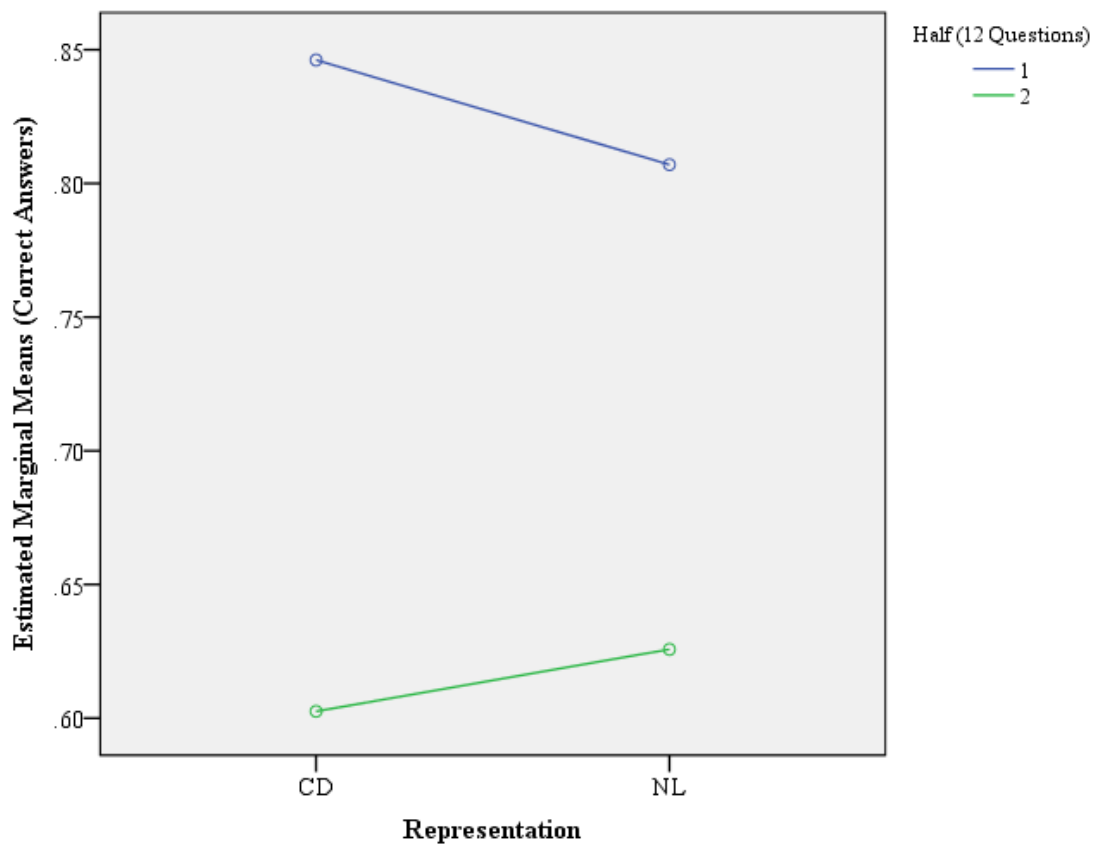


Figure 5.8 Graph of the interaction of CD and NL representations and correct answers for the two halves of the 24 questions

Although the performance of the CD group was not significantly different from that of the NL group, it is nevertheless possible that the constraint diagrams may not be effective because both groups were largely guessing the answers. As there are three answer options for each question, the level of chance performance on a question is 0.33, as shown by the dashed line in Figure 5.7. Many of the scores are substantially greater than this. It is possible to determine more formally whether each group performed better than chance by using a chi-squared test with 33.3% and 66.7% as the theoretical expectations of guessing correctly or incorrectly. Out of the 24 questions, four of the NL group's answers (Q14, Q18, Q19 and Q22) were not significantly different to chance and nine of the CD group's answers (Q1, Q8, Q11, Q14, Q15, Q16, Q18, Q19 and Q22) were not significantly different from chance, at $p < 0.002$. In this respect the CD group did not perform as well as the NL group.

One way to consider the relative impact of the two representations compared with other factors on the difficulty of giving answers is to determine the strength of the correlation between the proportions of correct answers for each group question by question. The greater the correlation the less likely that aspects which are specific to one or other representation are responsible for the level of performance. By using the Pearson Product Moment Correlation, the two variables were strongly correlated, $r(22)=0.601$ which is significant at $p<0.001$. In other words, the same questions are of comparable difficulty for both groups. Again this suggests that the performance of the two groups was not substantially different.

As a result, the CD group performed as well as the NL group despite the fact that it was their first time of being introduced to this notation.

5.5.2. Time Spent on each Question

This measurement is used to find out which notation would be faster in terms of learning, understanding and answering related questions. This could reflect their understanding of learning and using that notation. If using CD notation results in slowing the interpretation of a program specification, then this notation is worse. We believe that using CD notation will be worse than NL since it is a new notation and participants need to take time to interpret the question and the related case before answering, which will reflect their learning and understanding of such new notation.

Figure 5.9 shows the mean amount of time spent on each question by the two groups. The overall mean amount of time spent by the CD group is less than for the NL group across the questions. There appears to be a general trend for the proportion of time to increase from the earlier to the later questions, although the question-by-question variability increases in the second half of the questions. The mean (and SD) of the CD and NL groups over all the questions are 23.09 (16.84) and 28.73 (25.77) seconds, respectively.

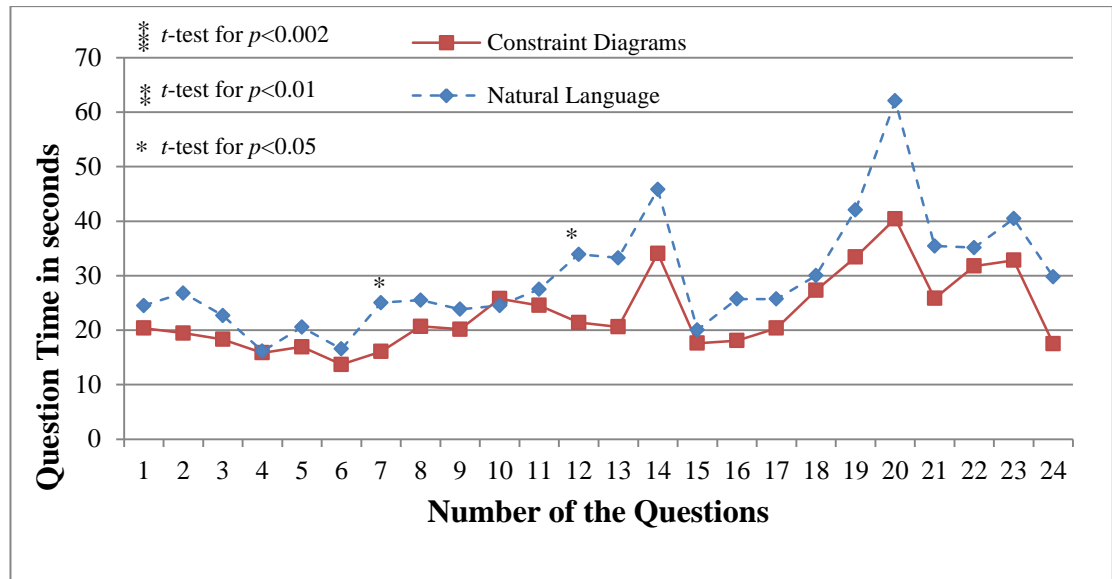


Figure 5.9 Graph of the average time for the two groups across the 24 questions

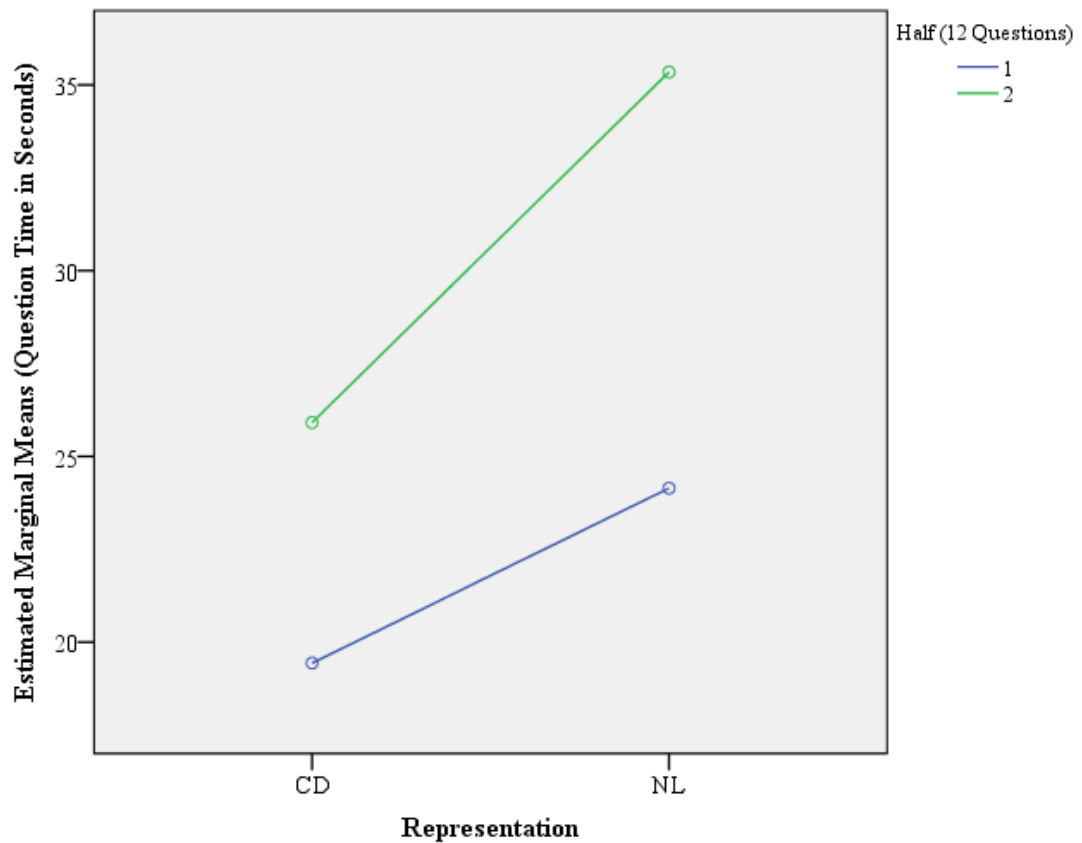


Figure 5.10 Graph of the interaction of CD and NL representations and the time spent on questions for the two halves of the 24 questions

There is no significant difference between the two groups, according to t tests (one-tailed) for $p < 0.002$. At $p < 0.05$, the difference between the groups that appeared in Q7 is $t(31) = 0.043$ and in Q12, $t(31) = 0.029$. As seen in Figure 5.9, in only one question

was the NL group a little faster than the CD. Overall, the CD group performed substantially better than the NL group.

The Pearson Product Moment Correlation showed that the two variables were strongly correlated, $r(22) = 0.901$, which is significant at $p < 0.000$. Both groups took a comparable amount of time to do the same questions, suggesting that the performance of the two groups was not substantially different. However, a Binomial test for 23 out of 24 questions being shorter, assuming equally probable that either would be shorter, has a probability of $p < 0.001$.

As a result, the CD group spent less time in answering the questions than the NL group. Thus, CD notation is better than NL notation in terms of fast learning.

5.5.3. Confidence Rating

This measurement was used to find out which notation would be more difficult to understand, which would affect their confidence in providing the answers. If using the CD notation resulted in having less confidence in answering rates than using NL notation, then NL notation is better, which is the hypothesis here since CD is a new notation and participants need to learn and understand such a new notation to be more confident in using it.

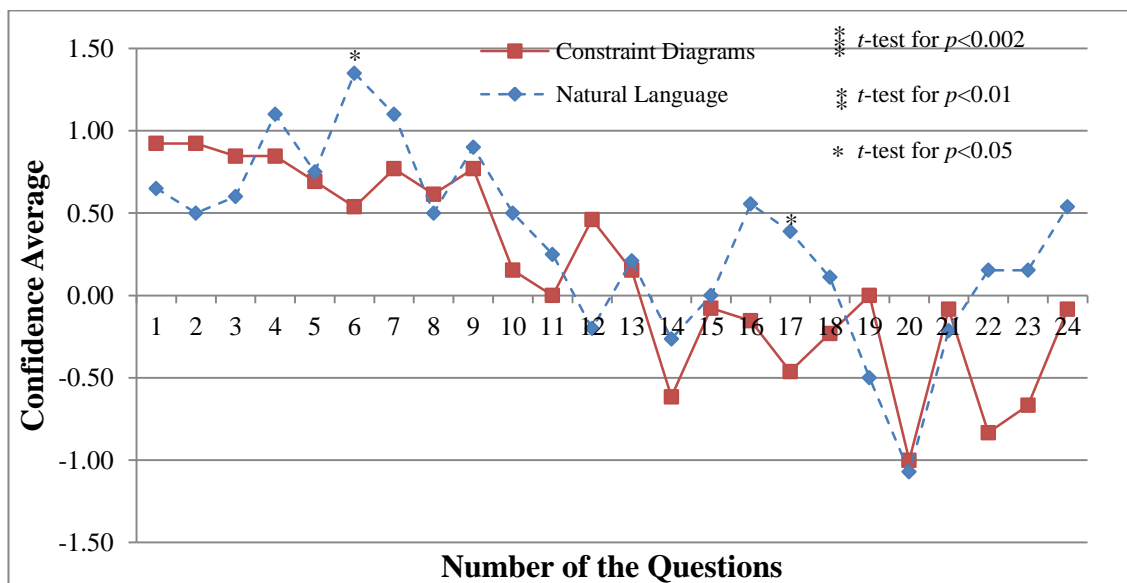


Figure 5.11 Graph of the confidence average for the two groups across the 24 questions

Participants answered the question on how hard they found the question by choosing one of five levels: very easy, easy, intermediate, difficult or very difficult. Their answer meant very confident, confident, neutral, not particularly confident and not at all confident. These levels of confidence rating were translated into numbers: 2, 1, 0, -1 and -2, respectively.

Figure 5.11 shows the mean of the level of confidence rating for each question for the two groups. The overall mean of the level of confidence rating by each group is different across the questions. There is a general trend for the level of confidence rating to decrease from the earlier to the later questions, although the question-by-question variability increases substantially in some parts of the questions. The mean (and SD) of the CD and NL groups over all of the questions are 0.16 (1.26) and 0.38 (1.17), respectively.

By using a mixed design ANOVA as shown in Figure 5.12, all the effects (the main within-subjects effect, the main between-subjects effect, and the interaction effect) are reported as not significant. For the main within-subjects effect of both halves of the questions, $F(1,30)=3.8$ for $p=0.06$ and for the main between-subjects effect for the two representations, $F(1,30)=0.868$ for $p=0.36$. Moreover, there is no significant interaction effect: $F(1,30)=3.09$ for $p=0.09$. Figure 5.12 indicates that the level of confidence in the first half for both groups was quite similar. However, in the second half the CD group was less confident while the NL group was very confident in the same half. It may be the case that CD effectively supports simple problems, or the case that CD is a new notation which needs more training. According to the significance results, there is no significant difference between the two halves and no significant difference between the two representations. Also, there are no interactions between representations and halves. As a conclusion, the CD notation is not worse than NL notation.

On one hand, by finding the average of the level of confidence rating values for each participant, we found most of the averages for the first half of the questions were greater than the averages for the second half. This is an indication that most of the participants in the CD and in NL groups were less confident in the second half than the first half. However, by using t test there are no significant differences at $p<0.002$.

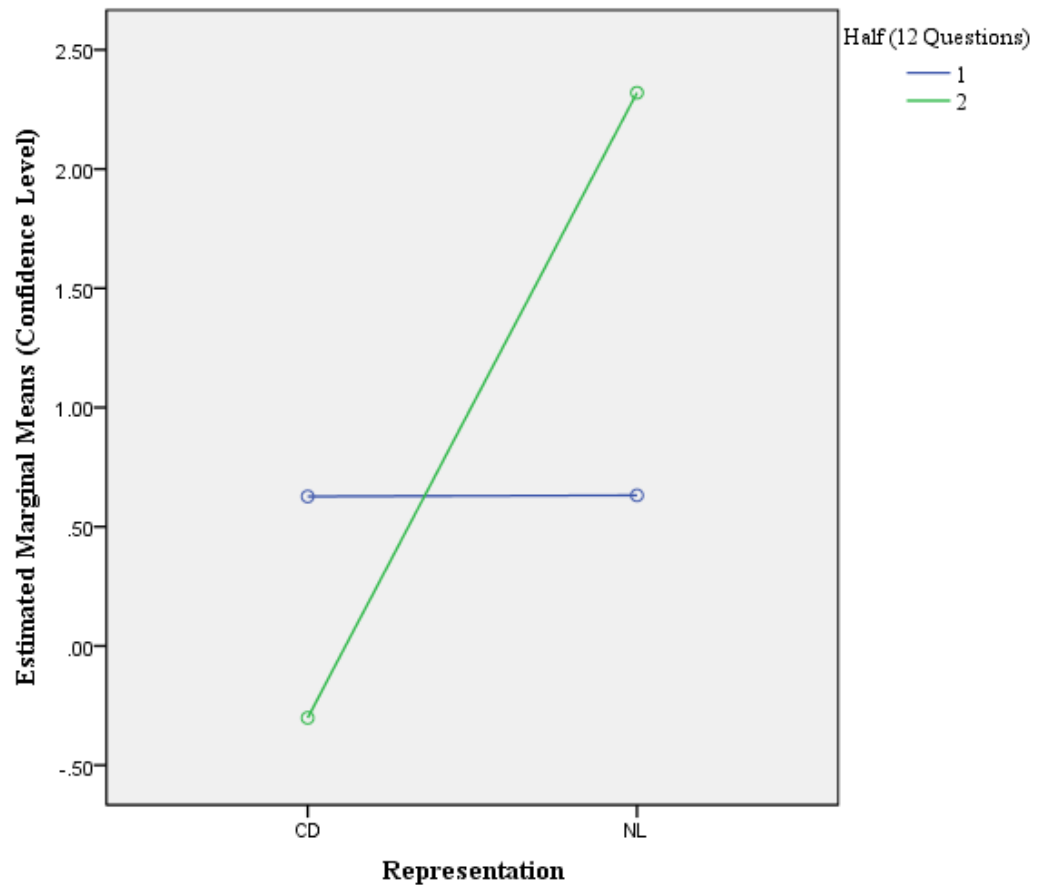


Figure 5.12 Graph of the interaction of CD and NL representations and the level of the confidence rating for the two halves of the 24 questions

On the other hand, we normalized the level of the confidence rating values for each participant by compare the level of the confidence rating of individual question with the overall average of the questions answered by that participant. Normalizing confidence try to take in the account the individuals bias of the participants and because we have quite a change between the level of the confidence rating and the normalized level of the confidence rating data it means it is important to normalize because basic rating between the two groups are not the same. We found out most of the averages of the first normalized half of questions are also greater than the averages of the second half which means the most of the participants in the CD and in NL groups are less confident in the second half. However, by using t test, we also did not find any significant differences at $p < 0.002$.

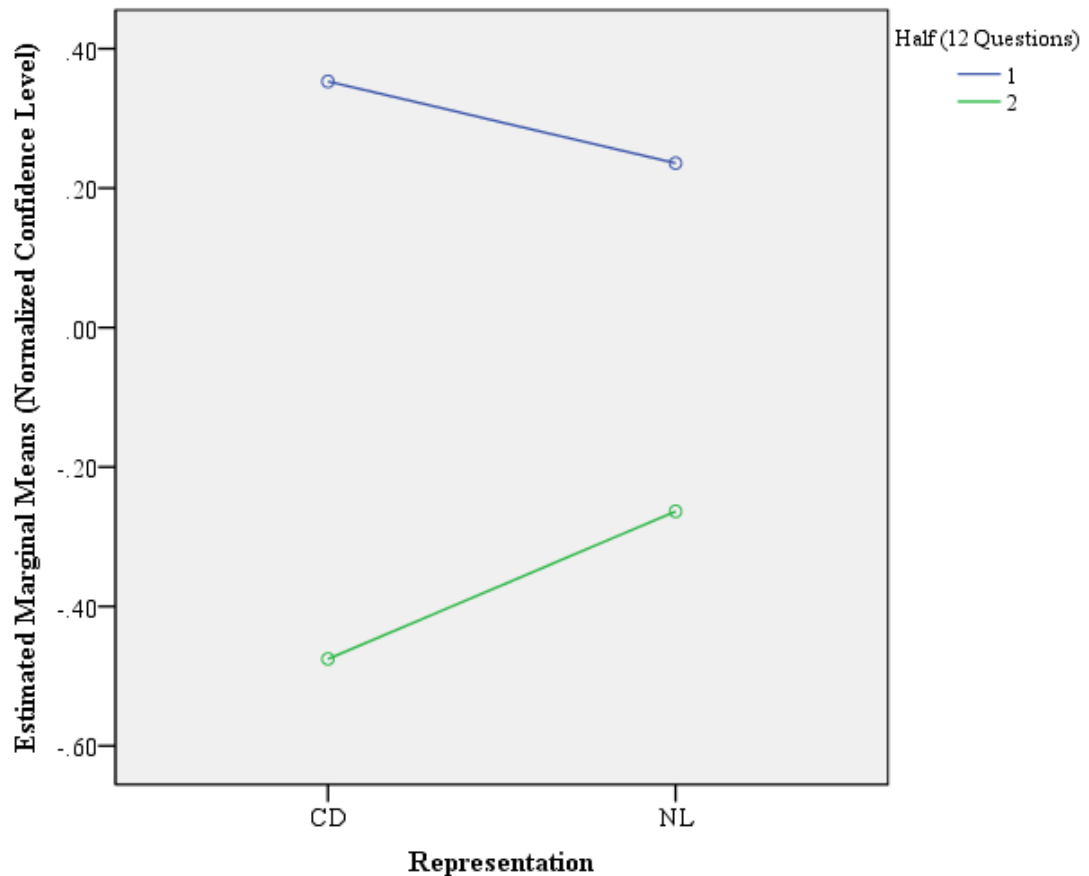


Figure 5.13 Graph of the interaction of CD and NL representations and normalized confidence rate the two halves of the 24 questions

To examine whether there is a change from start to end in confidence decline, a mixed design ANOVA is used. The interaction main effect between the two representations for the two halves is not statistically significant, $F(1,30)=2.19$ for $p=0.15$, and also it is not significant for main between-subjects effect for the representation: $F(1,30)=1.19$ for $p=0.28$. However, the main within-subjects effect is significant: $F(1,30)=35.68$ for $p<0.01$ and $\eta^2=0.543$. Figure 5.13 shows that the second half was worse than the first for both representations. For the CD group, they were more confident in the first half compared with the NL group. It may be the case that the CD notation effectively supports simple problems. However, in the second half the CD group were less confident than the other group. It may be the case that CD is a new notation and needs more training. In general, the confidence in the first half was higher than in the second half which indicates that there are no differences between the two representations and no differences between the two halves for each group. As a result, the CD notation is not worse than NL notation. By looking at the normalized confidence rate it actually shows a pattern more consistent with what we expected to find.

By examining each question individually, we find that there is no significant difference between the two groups in all but two of the questions, according to t tests (one-tailed) at $p < 0.002$. However, at $p < 0.05$, there are differences in Q6; $t(31) = 0.027$ and in Q17; $t(31) = 0.042$. The CD group achieved a level of confidence rating that was less than that of the NL group in 16 questions. In general, the CD group did not perform substantially better than the NL group.

By using the Pearson Product Moment Correlation, we found that the two variables were strongly correlated, $r(22) = 0.688$, which is significant at $p < 0.001$. The same questions were of comparable difficulty for both groups. As a result, the performance of the two groups was not substantially different.

As a result, the CD group were as confident as the NL group in answering, and thus they performed as well as the NL group.

5.5.4. Time Spent on each Example

This measurement is used to find out which notation needed less time to be understood and learnt. If using CD notation needs more time to learn how to interpret the specification for using it than NL notation, then this notation is worse than NL notation. We think that CD notation will be worse than NL which is the hypothesis here since it is a new notation and participants need to more time to study the new concepts represented in each example, which will reflect their learning and understanding of such a new notation.

Figure 5.13 shows the mean amount of time for each example for the two groups. The overall mean amount of time by each group is largely different across the examples. There appears to be a general trend for the amount of time to increase from the earlier to the later examples, although the example-by-example variability decreases substantially in some parts of the examples. The mean (and SD) of the CD and NL groups over all of the examples are 53.91 (57.60) and 33.03 (48.67) respectively.

For both halves of the questions, there is a significant main within-subjects effect of the two halves of the questions: $F(1,30) = 6.298$ for $p < 0.05$ and $\eta^2 = 0.174$. However, there is no significant main between-subjects effect of the two representations: $F(1,30) = 2.018$ for $p = 0.17$; and the interaction between the two representations for both halves shows that it is not significant: $F(1,30) = 1.242$ for $p = 0.27$. The graph (Figure 5.15) shows that

the CD group needed a longer training time in both halves compared with the NL group in general. Both CD and NL groups took more time in the second half compared with the first half. It may be the case that as it was a new notation it needed more training. Due to the lack of significant interaction effect, that there are no differences between the two representations and no differences between the two halves for each group. Thus, the CD notation is not worse than the NL notation.

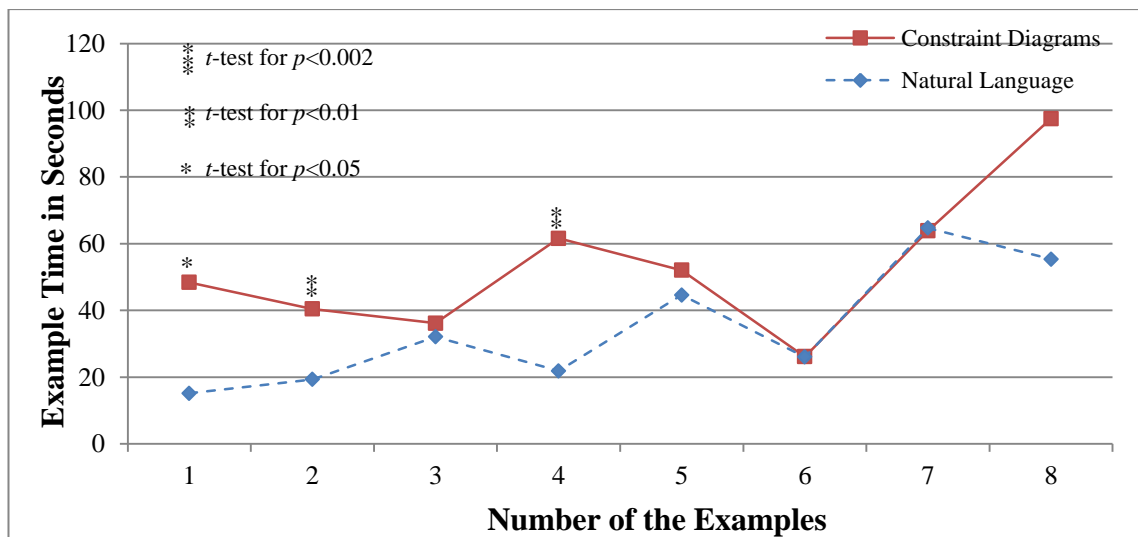


Figure 5.14 Graph of the example time average for the two groups across the 24 questions

By examining each question individually, we find that there is no significant difference between the two groups in all but two of the examples, according to t tests (one-tailed). Although by using t test, there are significant differences between the groups – in E1 it is significant, $t(31) = 0.018$ at $p < 0.05$, and similarly for E2 and E4, $t(31) = 0.006$ at $p < 0.01$ – there are no significant differences at $p < 0.002$. Overall, the CD group performed as well as the NL group.

Since the same examples were of comparable difficulty for both groups, the performance of the two groups is not substantially different. For the Pearson Product Moment Correlation, the two variables were strongly correlated, $r(6) = 0.607$, which is significant at $p < 0.05$. However, this correlation is not significant at $p < 0.001$.

As a result, there is no significant difference between the CD group and the NL group in terms of the time spent on the examples and thus they performed as well as the NL group.

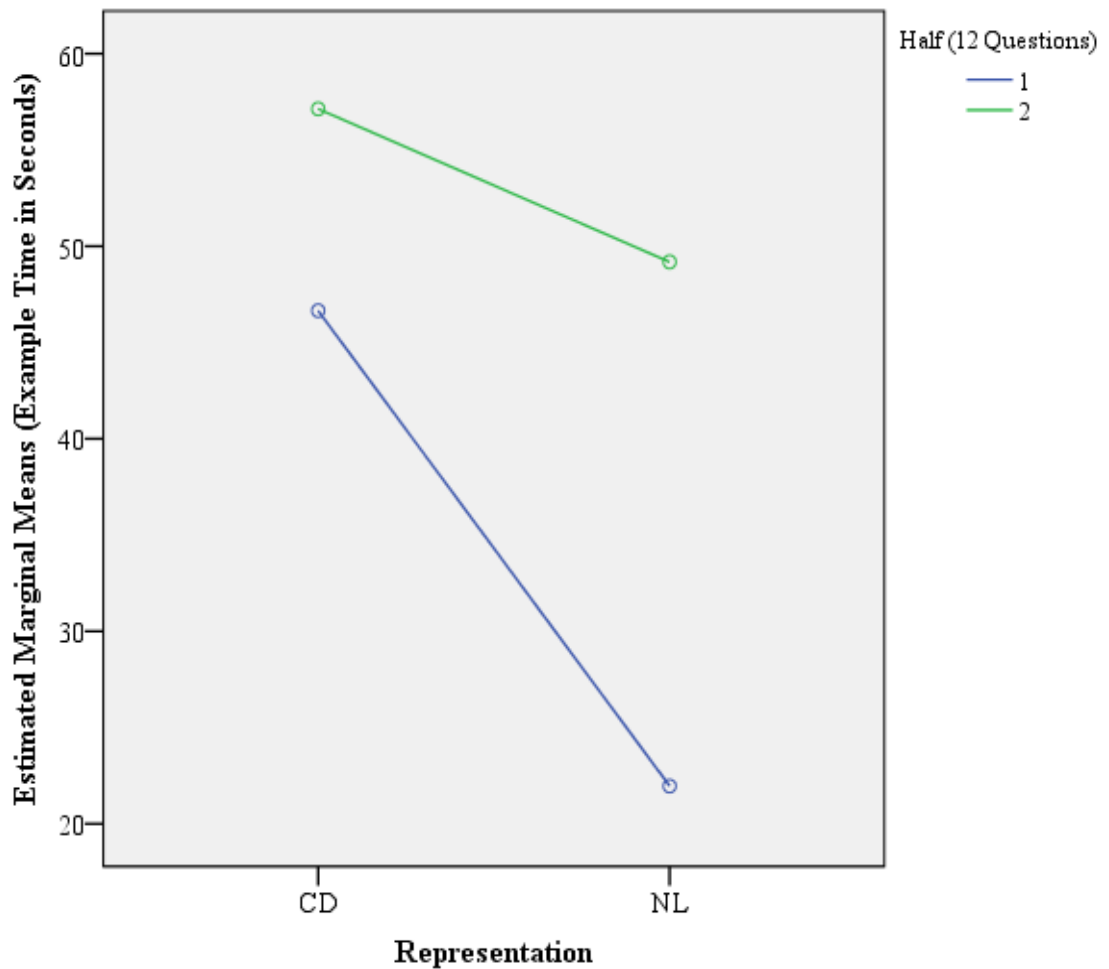


Figure 5.15 Graph of the interaction of CD and NL representations and the time spent on examples for the two halves of the 24 questions

5.5.5. Returns to Example

The participants in both the CD and NL groups did not choose to return to the examples, with the exception of one participant in the NL group and two in the CD group with very few returns. This is despite the fact that it was predicted that they would return to the examples more often, especially the CD group.

5.5.6. Relations between different measures

This subsection aims to find out whether there are any patterns of relations and any evidence of similarities between the two groups and between variables.

The Bonferroni correction is used here as well by dividing the conventional significance level which is 0.05 by the number of the tests which is 6, to find the adjusted significance level, which is ($p < 0.008$).

Table 5.1 The Pearson Product Moment Correlation between different measures for CD group

Pearson Product Moment Correlation	Correct Answers	Confidence Rating	Example Time
Question Time	-0.318	0.180	-0.028
Example Time	-0.283	0.661	
Confidence Rating	-0.380		

Legend: Level of significant for one-tailed, * $p < 0.008$ level for (df=11).

Table 5.2 The Pearson Product Moment Correlation between different measures for NL group

Pearson Product Moment Correlation	Correct Answers	Confidence Rating	Example Time
Question Time	-0.251	-0.208	0.642
Example Time	-0.412	-0.146	
Confidence Rating	-0.385		

Legend: Level of significant for one-tailed, * $p < 0.008$ level (df=18).

Table 5.1 shows the Pearson Product Moment Correlation value for the CD group across all the measurements: correct answer and question time, correct answer and example time, correct answer and confidence rating, confidence rating and question time, confidence rating and example time, and example time and question time. There are no significant relations between these measurements at $p < 0.008$.

Table 5.2 also shows Pearson Product Moment Correlation value for the NL group across all the previously mentioned measurements. There are no significant relations between these measurements at $p < 0.008$.

5.6. Discussion

The overall aim of this experiment was to evaluate the potential utility of CD for users who have relatively little experience at specifying programs by comparing how easily such users can learn CD notation and interpret specification expressions in comparison with using natural language (NL) expressions. It was predicted that participants using CD notation would (a) take longer over the training, (b) need more time to answer the questions, (c) be less confident about their answers, and (d) obtain lower correct interpretation scores, because they had no prior experience of CD notation. Although the CD group (a) spent more time on the training and (c) had less confidence, they (b) took less time to answer the questions, and (d) obtained equivalent correct interpretation scores to the NL group. Within-group analysis shows that there are no obvious classes of questions/concepts that one of the representations deals better with than the other. Although we expected to find that some questions might be more confusing for one of the representations than the other, there were no systematic differences. The number of

correct answers and the level of confidence might have decreased across time due to harder questions. Moreover, between-group analysis and item analysis showed that there was no particular question that was easier in one representation than in the other.

The experiment took the form of a web-based competition in which 33 participants were given instructions and training either on CD or equivalent NL specification expressions, and then responded to multiple-choice questions requiring the interpretation of expressions in their particular notation. Although we got positive results from this experiment, we cannot say that CD is a brilliant tool in every way because of the limitations of the experiment. Overall, it seemed that CD is an intuitive and expressive tool with unambiguous semantic notation, which supports Kent's claim (Kent, 1997). It is also easy to understand its interpretation when using CD to design software systems (Howse & Schuman, 2005). However, the proportion of correct answers and the confidence level on answering indicate that the interpretations were not always well-matched to their meaning (Stapleton & Delaney, 2008).

The difference between the two groups was rather too weak to be considered a substantial effect and the statistical results from this experiment are limited to considering the interpretation of specifications. The overall results (Table 5.3) show that the CD group were not any worse in terms of interpretation scores, which could be seen as encouraging. Since the variability of scores for within-group difference is smaller than those for the between-group difference, no real effect can be hidden.

The general methodology decisions might weaken the experiment. If using CD or NL to really try to understand specification programs, participants may adopt a different strategy. It is possible that the participants may simply have been using their general knowledge of the situation to answer the questions by reading the labels of the invariant/event. However, given that they studied the diagram typically for half a minute, it appears they were trying to interpret the diagram. In general, the average time spent on questions was less than half a minute. Since questions 19, 20, 22 and 23, which related to two concepts associated with these questions (concept of invariant and concept of event) took over half a minute, it seems participants took these questions seriously. These two concepts have labels that match the interpretation of the diagram. The matching label, in the case of interpreting complex diagrams, could be used as a guide for answering or could be avoided if it was interpreted as a trick.

Table 5.3 Summary of the Results

	Correct Answers	Time spent on Questions (in seconds)	Confidence Rate	Time spent on Examples (in seconds)
Overall CD mean (SD)	0.729 (0.445)	23.09 (16.84)	0.16 (1.26)	53.91 (57.60)
Overall NL mean (SD)	0.745 (0.436)	28.73 (25.77)	0.38 (1.17)	33.03 (48.67)
Chi –Test/ T-Test result for individual questions or examples between the two groups	There is a significant Difference Q12 ($p<0.002$) Q20 ($p<0.005$) proportion = 12questions for both groups.	No significant Difference at $p<0.002$. Q7, Q12 ($p<0.05$) Proportion = 1question in NL faster. (in CD, 23questions less time)	No significant Difference at $p<0.002$. Q6, Q17 ($p<0.05$) proportion = 16questions in CD (in CD, 8questions more confident)	No significant Difference at $p<0.002$. E2, E4 ($p<0.01$) E1 ($p<0.05$) proportion = 1example in CD (in CD, 1example less time)
Conclusion about Performance	CD is not worse than NL	CD is better than NL	CD is as well as NL	CD is as well as NL
Performance not above chance (i.e. they guessed)	4 questions in NL 9 questions in CD CD not as well as NL			
Correlation between the two groups	Significant at p <0.001	Significant at $p <0.001$	Significant at p <0.001	Not Significant at $p <0.001$
Overall Conclusion	Same questions/examples are comparably difficult Performance of both not substantially different.			

The drop-out rate was an indication of having problems with the representation or participants who might want to self-select the representation. If we allow the self-selection option, then we are biasing the comparison between the two groups. For fair comparison, participants were randomly assigned to one group despite what representation they would have preferred. Although we found that CDs are as effective as NL, it was clear that there was a difference between early and later drop-out especially in the CD group. If we assume that participants might have dropped out

because they did not want to interpret that representation, then – by comparing the representation preferences – CD is not worse than NL. Indeed, the drop-out rate might be independent of that representation. It might be the case that the dropout choice was self-selected due to the lack of familiarity with CD notation. It is more likely that participants who dropped out didn't like program specification in general or were not happy to think in a program specification way. Furthermore, it might be the case that the questions were getting harder and were not interesting. We do not know how the drop-out rate affected the results, but we know it has some limitations. We expected a higher drop-out rate in the CD group because of the unfamiliarity of CD. However, there were an equal number of drop-outs in both groups. This indicated that the drop-out was either independent of the representation or some degree of representation effects.

Many experimental design phases were created. It is a complicated design and we do not know how these decisions affected the performance. Basic concepts were covered in this experiment because the aim of the experiment was to examine how easily novices would understand this simple and effective notation called constraint diagrams (Kent 1997; Howse and Schuman 2005). We have not included the strand syntax because we examined simple spider relations and strand is a more complex kind of spider relations. However, if we include strand, we might find relations between spiders is a difficulty.

By having an online experiment, more participants could be involved and they were allowed to choose the best time to start. However, they might exclude themselves because they faced a bad online connection or a rough day. Moreover, they might not take this experiment seriously because it did not occur in a classroom and it was not related to their study. In fact they might be encouraged to complete the experiment if they saw other participants working on it at the same time. As it was a training experiment, participants had to both learn and perform a task at the same time. Furthermore, as it was a competition experiment, it might be the competition aspects that affected their performance in order to try and win. Immediate feedback had a valuable effect on their learning. However, a lack of detailed feedback to participants and a lack of understanding on our part of what the participants faced were drawbacks. It may also be a factor that prompted them to drop out. The limitation of the confidence rate is the fact that it is only a guide which is used as an absolute measure to use to compare the two groups. Returning to examples was not a useful measure. As it was a competition and the time taken was recorded, they might have decided to save time by

not returning to examples. Or they might understand the notation well enough just from one reading.

The findings might be more significant if an experienced CD group had been compared with an experienced NL group. Indeed the group is experienced in NL but not in using NL for specifying programs. Choosing to use multiple choice rather than free form responses could affect the results' value. They only interpreted simple diagrams with building models. Thus, this experiment has a limited ecological validity because participants were asked to respond to multiple-choice questions whereas a program specification task is more about involving participants. The experiment did not really capture the sort of interactions and understanding of CD notation that would be required in a real-world setting – in particular the use of multiple choice rather than more open-ended responses.

In the next experiment, we will examine the use of CD notation to construct a specification of a system and we will compare it to the use of NL notation to construct the same specification of the same system. Constructing CDs will be more interesting and relevant than repeating this experiment, to classify participants' visual ability and to try to put them into two different groups.

Chapter 6 Experiment 2: Construction of Constraint Diagrams

This experiment attempts to evaluate the efficacy of CD by comparing two different groups to check how easily CD notation can be used for constructing specification in diagrammatic expressions compared with NL notation. Section 6.1 introduces the experiment. Section 6.2 represents the experimental design. Section 6.3 explains the pilot experiments. Section 6.4 shows the results of the experiment. Finally, section 6.6 is the discussion.

6.1. Introduction

This chapter describes a construction-tasks investigation to explore the usability of CD as a program specification language by focusing on three different cognitive activities for learning: thinking, reasoning, and reflecting. This experiment took the form of a factorial experiment in which 20 participants were asked to construct the specification from one representation into another. They were given instructions and training either on CD or in equivalent NL specification expressions, and they then responded to construction questions requiring the generation of expressions in their particular notation. It was predicted that participants using CD would take longer over the training, obtain a lower number of correct construction scores, need more returns to examples, and take longer over examples they returned to because they had no prior experience of CD notation.

This empirical study, which is the second step in the empirical evaluation of this notation, is about examining participants' ability to understand particularly the notation usage in real life problems by getting them to use a representation either in linguistic or graphical modalities. It is aimed at rigorously evaluating the effectiveness of CD by detecting any error of interpretation and finding out whether differences in the nature of representation make a big difference in construction, whether learning and using CD for constructing is going to be a major task compared with other approaches, and whether, after training, CD will be as effective as the comparator notation. In order to answer the question of whether CD could have higher ecological validity because participants will be involved in this experiment to capture the sort of interactions and understanding of CD notation that would be required in a real-world setting, we examined their

interpretation with basic concepts and simple questions to find out if there was a fundamental difference between CD and NL, which had not been done before. Thus, we performed this experiment to identify the benefits of using CD in specifying programs. In this experiment, we tried to answer some questions such as whether is it possible to translate and generate CD diagrams or Formal-NL (FNL) statements under certain circumstances, whether translating a statement written in ordinary-NL to another representation such as CD or FNL is easy, and which of CD or Formal-NL is easiest to build a model from, using ordinary-NL.

6.2. Experimental Design

Just as we did in experiment 1, we will now consider: (a) the design of the experiment, (b) the approach to familiarizing people with constraint diagrams, (c) the measures that we will use as evidence, (d) the predictions, (e) the design of the examples and the questions, (f) the concepts, and (g) the appropriate number of questions.

(a) For the same reasons we provided for experiment 1, a between-participants design was adopted with two separate groups and two different representations: CD and NL. Two different versions of the same material, questions and examples were created, which were informationally equivalent (Larkin & Simon, 1987). There were many possible designs for this experiment. (1) An abstract description would be provided and participants would be asked to construct the model/specification by precisely translating to CD/NL. CD is a formal notation and there would be a list of terms that could be used. Although NL is an informal notation, it could be considered as a semi-formal notation; we called it Formal Syntactical Language (FSL), when a list of terms that could be used was provided to describe the problem in terms of those. (2) They would be asked to construct the model/specification by precisely translating from CD to NL and vice versa. (3) They would be asked to construct the diagrams by translating from NL to CD and during the second half they would be asked to construct the expressions by translating from CD to NL, which is a complicated design. Moreover, grouping participants has many different possibilities, such as two groups – CD and FSL – or three groups, CD, FSL and hybrid (mixed/combined). We chose two groups and adopted design (1); they would randomly be assigned to the CD group or the NL group to construct either expressions or diagrams.

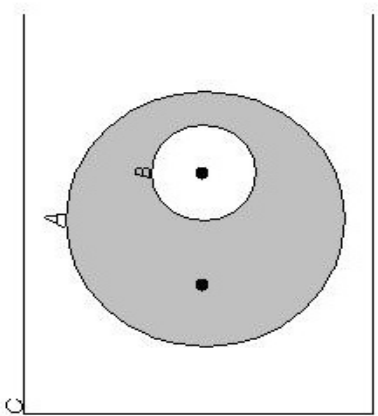
(b) In spite of the fact that there are individual differences in cognitive styles, “It is possible to compensate for these differences by training” (Blackwell, 1997), which is considered to be similar to such arguments in other studies (Frandsen & Holder, 1969). Participants gained benefits in experiment 1 when they had been trained on constraint diagrams in terms of familiarizing them with the new notation, CD. Thus, this would allow a fair comparison between these two different representations. Since this experiment is about modelling with CD and FSL, participants would be trained on how to use our tool, the editor software, by familiarizing them with the interface and the procedure to help when starting with the material. They would be introduced to an example to familiarize them with how training examples are shown in this editor, as shown in Appendix B. Each training example was used to define a new concept. The training was a step by step approach to teach them how they were supposed to read the examples. Then they would have a pair of questions to represent the same situation. However, part A is an informal description that we normally use and part B is formal and specific, as shown in Appendix B. They would be asked to reproduce the given diagram or recreate the given sentence/expression that appeared in the two parts of the question. In both parts of this training question, as shown in Figures 6.1 and 6.2 for CD and Figures 6.3 and 6.4 for FSL, a step by step instruction on how to use the editor and how to construct diagrams/expressions was provided, followed by a button to ensure that they read every instruction. In the case of CD, we gave them an empty-labelled set with a labelled subset. These two sets have a labelled spider and non-labelled spider, respectively, as shown in Appendix B. All these diagrams were included in an invariant. After that, the correct answer would be provided. These steps were carried out to familiarize them with constructing using the editor.

Then we familiarized them with the concepts themselves by introducing a training example that defines a concept by a brief description. We then examined their understanding by generating diagram/expressions by three related questions. Each question had two parts: part A and part B. The two parts represent the same description. However, part A is a more informal NL specification which is represented by an everyday description while part B is a more specific NL specification.

c) In order to evaluate how effective the different notations were, we measured how accurately and how quickly they were used in training and in translating tasks. The measurements used were: time spent on the training examples, time taken in returning to

Training Question - Part A:

A circle, called B, is contained in another circle, called A. The area of A outside of B is shaded. The area of intersection between A and B is not. There is a dot in B and also a dot in A which is not in B. A single-open-rectangle, called C, contains these two circles. The produced diagram should look like this:



Please try to reproduce this diagram by using the on-screen editor.

The question will be:

From the given statement, construct a diagram that asserts this situation.

•

•

Example

Next

Figure 6.1 Evaluating Methods of Program Specification: Constructing CD – Training Question: Part A

Training Question - Part B:

There is an element in a set called A and there is another element in a set called B . The intersection of A and B is B . If the element is in A but not in B , then it is the only element there. All of this is defined in a class called C .

The produced diagram should look like this:

From the given statement, construct a diagram that asserts this situation.

□

○

●

◐

↶

*

—

;

□

≡

Example

Next

Figure 6.2Evaluating Methods of Program Specification: Constructing CD – Training Question: Part B

Training Question - Part A:

A group, called B, is contained in another group, called A. The members of B are already members of A. However, if there is a member in A but not in B, then this is the only member there. B can have many members. An invariant, called C, contains these definitions.

The produced formal expression should look like this:

Is-Invariant(C)

Is-Set(A)

Is-Set(B)

Subset(B,A)

No of-Elements(B, ≥ 1)

No of-Elements(A, = 1)

Please try to reproduce this formal expression by using the on-screen editor.

The question will be:

From the given statement, construct a formal expression that asserts this situation.

Is-Type()

Is-Set()

Disjoint()

Is-Element()

All-Elements()

Of-Type()

Some-Elements()

Element-in-Set()

Is-Relation()

No.of-Elements()

Same-Element()

Intersect()

Is-Invariant()

Subset()

Is-Event()

Example

Next

Figure 6.3 Evaluating Methods of Program Specification: Constructing FSL – Training Question: Part A

Training Question - Part B:

There is an element in a set called A and there is another element in a set called B . The intersection of A and B is B . If the element is in A but not in B , then it is the only element there. All of this is defined in a class called C .

The produced formal expression should look like this:

Is-Invariant(C)

Is-Set(A)

Is-Set(B)

Subset(B, A)

No.of-Elements($B \geq 1$)

No.of-Elements($A = 1$)

Please try to reproduce this formal expression by using the on-screen editor.

The question will be:

From the given statement, construct a formal expression that asserts this situation.

Is-Type()

Is-Set()

Is-Element()

Of-Type()

Element-in-Set()

No.of-Elements()

Intersect()

Subset()

Disjoint()

All-Elements()

Some-Elements()

Is-Relation()

Same-Element()

Is-Invariant()

Is-Event()

Example

Next

Figure 6.4 Evaluating Methods of Program Specification: Constructing FSL – Training Question: Part B

examples, time taken as initial thinking time, percentage of correct answers, percentage of productivity (created objects), percentage of corrected objects, percentage of accurate configurations, steps rate and number of returns to the examples.

(d) It was predicted that FSL would be better because it is NL-based so it is familiar and there is no need to learn a new notation before working on the tasks. Given the complexity of CD, the experimental hypothesis states that participants given the CD notation would obtain a lower number of correct answers proportionally in a longer training time with many returns to examples. This experiment has the version of the specification (NL-CD or NL-FSL) as the independent variable; whereas correct answers, initial thinking time, steps, productivity, configuration, example time and returning to examples are the dependent variables.

(e) For the same reasons that we provided for experiment 1, the domain used for examples was different from the one used for questions. Each example introduced a concept was provided from a domain called "Video Rental Service" while the questions' domain was called "Patient Record System". To recap, there were two design options: (1) an example followed by three questions or (2) all examples followed by all questions. The benefits of (1) was to incrementally construct knowledge by understanding each concept individually, which would help us to identify the difficult concepts. In contrast, (2) requires quite a heavy load of learning and remembering. We adopted (1) to allow learning each concept by elaborating on it. We believe that the questions and the domains are realistic to real world tasks.

(f) This experiment has seven concepts covered in the experiment: Sets and Types, Members of sets, Set relationships, Relationships, Spiders, Invariant and an Event Specification. Every concept was explained using an example that was followed by three related pairs of questions.

(g) A pilot experiment, which was a one-to-one experiment between the participants and the experimenter, was conducted in order to ascertain the level of difficulty and the number of questions. We concluded that 21 questions would be an appropriate number of questions.

6.3. Pilot Experiments

We had a small number of participants for these pilots. We first conducted a PowerPoint mock-up to simulate the proposed design. From this mock-up we selected the relative components and we tested the number of the questions required. After that, we conducted a java application to simulate the experiment itself. We ran a one-to-one pilot experiment, with four participants, to help the experimenter to focus on how participants interacted with the editor, how reliable the software was, and how adequate the material was. We found that the constructing activity took more time than the interpretation activity, and it was harder than the interpretation activity. Thus we concluded that the number of questions should be reduced from 24 to 21 questions and also decided to divide the experiment into two different times to allow participants to be more active.

For the pilot software design, we did not allow two similar shapes to have the same label. However, due to the complexity of the diagrams used and as a result of these pilots we found that we had to allow two similar shapes to have the same label, which helped participants to be more productive and to support the implicit reading tree of separating a diagram into multiple diagrams for easiness and to avoid ambiguity. For example in Figure 3.5 we have three circular shapes (contours) with the same label *Patient*. These three contours represent the same set, but they were repeated to provide more information and to help in keeping the diagram simple.

6.4. Experiment

6.4.1. Method

This experiment is another empirical study on learning constraint diagrams which is an extension to the previous one. This approach was chosen to get more fine-grain details of what may cause problems in the CD representation. Participants were randomly divided into two groups: (1) a group to create constraint diagrams, and (2) a group to create formal natural language statements. It was conducted at the informatics department lab at the University of Sussex where participants used the lab computers to participate in the experiment. They were trained on using the software and then they were introduced to training examples that, individually, were followed by three pairs of questions. As previously mentioned, each question had two parts: A and B. The data is collected for two purposes: to check the hypothesis that CDs are easy to learn and to

use, and to check for any unusual strategy a group may adopt. After analysing the data, we could check whether CD construction would be easier than NL or not.

This experiment has a high ecological validity because it is realistic and very near to a real life task. The experiment will help in ascertaining whether a statement written in a representation can be translated to another easily with the same meaning, and whether it is possible to generate CD or a semi-formal statement of a certain circumstance. We need to measure how accurately and quickly they can be trained and then let them translate. Then we scored the diagram/statement to see how accurate they were and we checked if there were missing objects and how long it took them to answer. The results depended on the independent variables such as CD, NL and the complexity of problem, and on the dependent variables which we used as measurements, such as the number of correct answers, the number of correct objects, the number of created objects, the number of accurate configurations, the number of steps, the time spent on the initial thinking, the time taken in training and learning examples, the time taken in training for new examples, the time taken in training for returned-to examples, and the number of returns to the examples.

Indeed, the environment of constructing diagrams/statements has a tool-bar to use, and a description of the expression/diagrams that we have to construct. We developed an editor for generating both diagrammatic and textual expressions. Also, we encoded program specification statements in four different representations; CD, informal-NL, precise-NL, and NL-based FSL. Despite the fact that developing software is a time-consuming method, it is a helpful tool to accurately record time and numbers of tries at answering, and to capture the diagrams without any requirement for hand-drawing or for drawing fast. We decided to let them drag and drop instead of drawing because drawing may cause ambiguity, so instead of drawing a full circle they might draw an illegal circle such as in Figure 6.1. By dragging and dropping, we avoided any problem of unclear drawing and we could easily judge if an answer was right or wrong. Moreover, dragging and dropping forced the NL group to use a limited number of words to construct the expressions, which was easier to analyse and made the mode interaction of NL similar to the mode interaction of CD. For example, in the CD version, shading represents a set that may be empty or has a specific number of elements, and a non-shaded contour represents a non-empty set. By matching this to the NL version, No.of-Elements (setName,#,=) represents a set labelled with setName

which may be empty (if # equals 0) or has a specific number of elements equal to # value, and No.of-Elements (setName,#, \geq) represents a non-empty set.

To get from the issue of how to automatically assess the diagram, we can provide feedback by showing them both correct and incorrect answers for both versions. The feedback was generated by the author from the editor itself as a way to assess the functionality of the editor. As a result, participants saw diagrams or expressions that were similar to what they could generate. We recorded what they did and scored everything automatically, except for accurate configuration measurement which was done manually. The scoring schema was very difficult to develop; we developed a scoring scheme by defining dimensions of marking to be correct such as correct objects, created objects, and configuration. For configuration reasons, we identified diagrams as legal diagrams, illegal diagrams, correct diagrams, incorrect diagrams, and partially correct diagrams. These demanded the collection of lots of information from the process of constructing by the use of log files to track the order of shapes, the number of the shapes, the name of the shape, the kind of shape, the type of action performed (resize, labelling, moving, select, delete, and undo), the number of actions performed, the time taken to perform every single action, the location of the shape, and the size of the shape. Moreover, we captured the final generated answer in a snapshot.

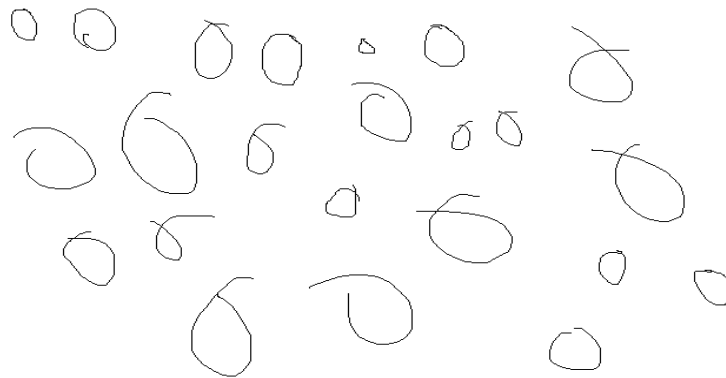


Figure 6.5 Examples of hand-drawn circles

6.4.2. Subjects

The subjects were undergraduate and postgraduate informatics students at the University of Sussex. Since this experiment aimed to study the potential of CD, not the modelling techniques, we used participants with some knowledge of modelling techniques such as informatics students. Our target was to concentrate on the notation

itself rather than on the modelling. They voluntarily participated in the experiment for £15 cash. There were twenty participants in total, divided randomly into two groups of ten. All of them completed the experiment without any withdrawals. In general, there were a smaller proportion of female participants, but the ratio of male to female was the same in the two groups: 8:2. They were asked to provide their Sussex University email, and their personal information. Moreover, they were asked to read the experiment instructions.

6.4.3. Materials

Participants had to study seven concepts provided by training examples. Each training example was followed by three questions. Figures 6.6 and 6.7 are screen-snapshots from the experiment for CD and NL versions respectively. Each represents example 7 (an Event Specification) in a different representation. An example has a title indicating the introduced concept, diagrams or statements depending on the version of the representation used, a description of the diagrams or the statements, and the *Next* Button.

Figures 6.8 and 6.9 represent a question (part A) which relates to example 7. The numbering of the questions was randomly assigned for the same training example. A question about *UpdatingPatientRecord* was assigned one time as question 19 and another time as question 20 or 21. The three questions related to example 7 were only assigned as 19, 20, or 21 randomly. The part A question had a very informal description used in everyday life. On the other hand, Figures 6.10 and 6.11 represent part B of the same question as part A, but with more details to give a precise description. Both part A and part B had the *Example* button to return to the training example in case more learning was needed and the *Next* button to go to the next related question. The *Next* button was not enabled until at least one button to generate a diagram or an expression had been pressed. Figures 6.12 and 6.13 present part B after pressing the *Next* button which produced a pop-up window with the expected answer.

Example 7: An Event Specification

There is an invariant called *VM*. This invariant has the definition of *Member* set and the *Info* relation as shown below:

```

Is-Invariant(VM)
Is-Type(W)
Is-Set(Member)
Or-Type(Member M)
No-Of-Elements(Member ≥ 0)
All-Elements(m)
Element-In-Set(m, Member)
Is-Type(I)
Some-Elements(I)
Is-Relation(Info, m, I)
        
```

Based on this invariant there is an event (operation) called *VM/NewMember(m, i)*. The invariant name, *VM*, which appears in the event's name is a reference to hidden information that previously defined. This event is used to register a new member, *m*, with information *i*. A specification of an event has 2 conditions; one condition (called pre-condition) is used to show things before changes such as initiated members of sets and the other condition (called post-condition) is used to show what has changed. The pre-condition (before applying the event) ensures that its input-argument, *i*, has type *I*, and that the other input-argument, *m*, is an identifier of type *M* which is not in *Member*. In the post-condition(after applying that event), *m* is associated with *i* by a relation called *Info* (Information) and it is in *Member*. Dashed names denote values that are changed- but only minimal changes are shown here; there is no need to say that other elements of *Member* and their associated *Info* values remain the same, because of the convention that "the rest stays unchanged".

The following formal expression represents this statement.

```

Is-Event(VM/NewMember)
Pre-Condition:
Is-Type(I)
Is-Element(i)
Or-Type(I, I)
Is-Type(W)
Is-Set(Member)
Or-Type(Member M)
No-Of-Elements(Member ≥ 0)
Is-Element(m)
Or-Type(m, M)
Post-Condition:
Element-In-Set(m, Member)
Is-Relation(Info, m, I)
        
```

Definition:
 Is-Event(C/E)
 Pre-Condition:
 Post-Condition:

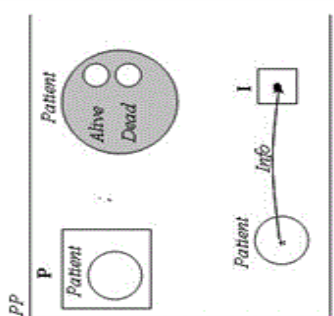
This expression represents an event called *C/E* which specifies some allowable change-of-state for objects. Thus it always involves a POST-CONDITION, and it may include an optional PRE-CONDITION as well. The ! is used to separate the Event and the invariant names.

Example Next

Figure 6.7 A snapshot of training example 7 using NL


Question 21 - Part A:

Given a concept called *PP*:



Current information for a patient can be updated to some new value.

From the given concept and statement, construct a diagram that asserts this situation.



Next

Example

Figure 6.8 A snapshot of “UpdatePatientRecord” Event (Question Part-A) using CD

Question 20 - Part A:

Given a concept called *PP*:

Is-Invariant(*PP*)
Is-Type(*P*)
Is-Set(*Patient*)
Or-Type(*Patient*, *P*)
Is-Set(*Alive*)
Is-Set(*Dead*)
Subset(*Alive*, *Patient*)
Subset(*Dead*, *Patient*)
Disjoint(*Alive*, *Dead*)
No.of-Elements(*Patient*, =, 0)
No.of-Elements(*Alive*, ≥, 0)
No.of-Elements(*Dead*, ≥, 0)
All-Elements(*p*)
Element-in-Set(*p*, *Patient*)
Some-Elements(*l*)
Or-Type(*l*, *l*)
Is-Relation(*Info*: *p*, *l*)

Current information for a patient can be updated to some new value.

From the given statement, construct a formal expression that asserts this situation.

Is-Type()

Is-Set()

Is-Element()

Of-Type()

Element-in-Set()

No.of-Elements()

Intersect()

Subset()

Disjoint()

All-Elements()

Some-Elements()

Is-Relation()

Same-Element()

Is-Invariant()

Is-Event()

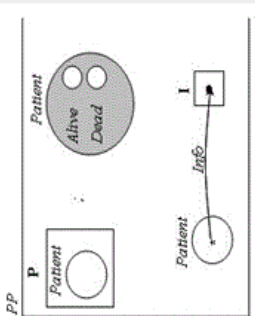
Example

Next

Figure 6.9 A snapshot of “UpdatePatientRecord” Event (Question Part-A) using NL


Question 21 - Part B:

Given an invariant called *PP*:



An event called '*UpdatePatientInformation(p,i)*' of class called *PP* can be used to update the current *Information* for a patient *p* to some new value *I* if and only if the current value of *Information* differs from the new value. Both values are of type *I*.

From the given invariant and statement, construct a diagram that asserts this situation.



Next

Example

Figure 6.10 A snapshot of "UpdatePatientRecord" Event (Question Part-B) using CD

Question 20 - Part B:

Given an invariant called *PP*:

Is-Invariant(*PP*)
Is-Type(*P*)
Is-Set(*Patient*)
Of-Type(*Patient*,*P*)
Is-Set(*Alive*)
Is-Set(*Dead*)
Subset(*Alive*,*Patient*)
Subset(*Dead*,*Patient*)
Disjoint(*Alive*,*Dead*)
No.of-Elements(*Patient*, = 0)
No.of-Elements(*Alive*, ≥ 0)
No.of-Elements(*Dead*, ≥ 0)
All-Elements(*p*)
Element-in-Set(*p*,*Patient*)
Some-Elements(*i*)
Of-Type(*i*,*I*)
Is-Relation(*Info*:*p*,*i*)

An event called '*UpdatePatientInformation(p,i)*' of class called *PP* can be used to update the current *Information* for a patient *p* to some new value *i* if any only if the current value of *Information* differs from the new value. Both values are of type *I*.

From the given statement, construct a formal expression that asserts this situation.

Is-Type()

Is-Set()

Is-Element()

Of-Type()

Element-in-Set()

No.of-Elements()

Intersect()

Subset()

Disjoint()

All-Elements()

Some-Elements()

Is-Relation()

Same-Element()

Is-Invariant()

Is-Event()

Example

Next

Figure 6.11 A snapshot of “UpdatePatientRecord” Event (Question Part-B) using NL

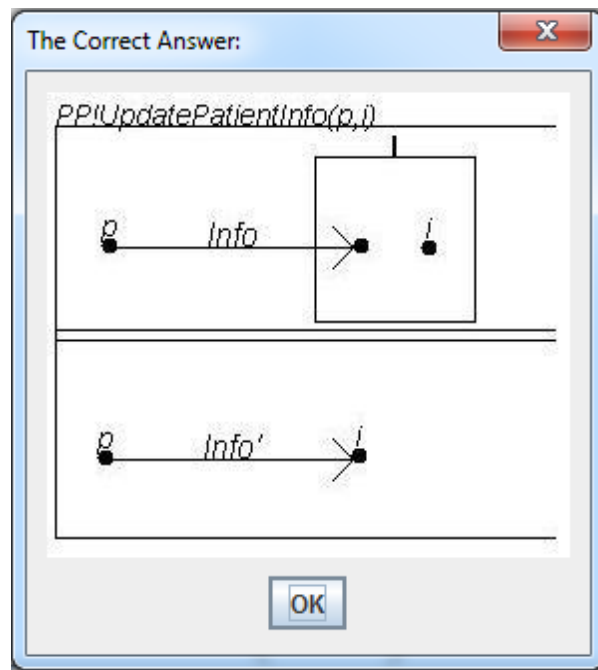


Figure 6.12 A snapshot of the correct answer of “UpdatePatientRecord” Event Question using CD

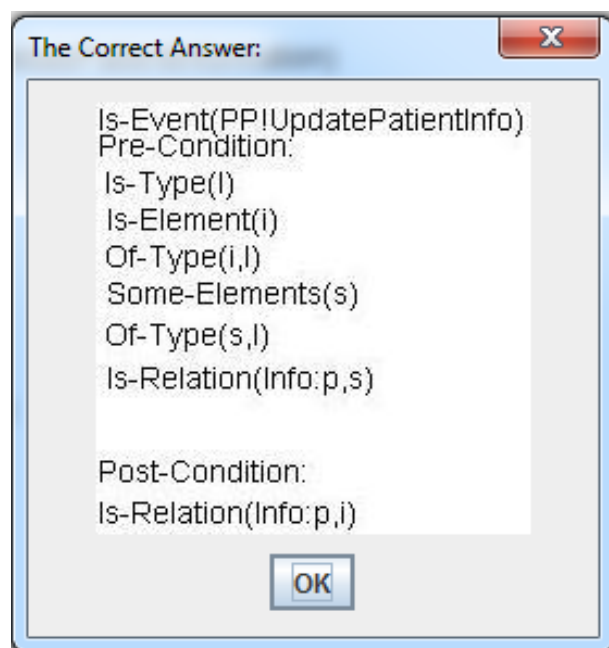


Figure 6.13 A snapshot of the correct answer of "UpdatePatientRecord" Event Question using NL

A full range of all the examples and questions used in this experiment is in Appendix B. The next section will explore the results.

6.5. Results

Overall, it was predicted that the participants in the CD group would find reflecting on the learnt concepts and constructing diagrams to answer questions in the domain harder. It was predicted that the CD group would have more incorrect answers to the questions,

and perform more steps. Similarly, it was expected that the amount of time spent studying the examples would be longer for the CD group. For the rest of this section we will, firstly, look at the results of analysing correct answers which required analysing the percentages of correct objects, created objects and the configuration. Secondly, we will find the number of steps needed to accomplish tasks. Thirdly, we will examine the number of returns to the training examples. Fourthly, we will evaluate the time spent on each example, whether it is a new example or a returned-to example. Finally, we will investigate the initial thinking time.

For the reasons explained in Chapter 5, the Bonferroni test was used as needed for this experiment to adjust the significance levels using the number of comparisons because several tests are done on the same data. By dividing the conventional significance level which is 0.05 by the number of the tests which is 21, the adjusted significance level is ($p < 0.002$) which applies to all the t tests. It is important to achieve the adjusted significance level for the t tests to be counted as significant.

6.5.1. Correct Answers

This measurement is used to find out which notation could provide more accurate answers that reflect their understanding of using that notation. If using CD notation to produce the specification will produce more correct answers than using NL, then CD notation will be better than NL. We believe that CD notation will be worse than NL because it is a new notation and participants need to learn it and understand it.

An answer is correct if and only if the number of correct objects (in section 6.5.1.1), the number of created objects (in section 6.5.1.2), and their configurations (in section 6.5.1.3) are correct.

Figure 6.14 shows the mean of correct answers for each question for the two groups. The overall mean number of correct answers by group is largely similar across the questions. There appears to be a general trend for the proportion of correct answers to decrease from the earlier to the later questions. The mean (and SD) of the CD and NL groups over all of the questions were 0.307 (0.461) and 0.157 (0.364) respectively.

If we calculate the mean of correct objects and configuration without the created objects, we found the mean of correct answers for each question for the two groups. The overall mean number of correct answers by group is largely similar across the questions.

There appears to be a general trend for the proportion of correct answers to decrease from the earlier to the later questions. The mean (and SD) of the CD and NL groups over all of the questions were 0.871 (0.126) and 0.834 (0.121) respectively.

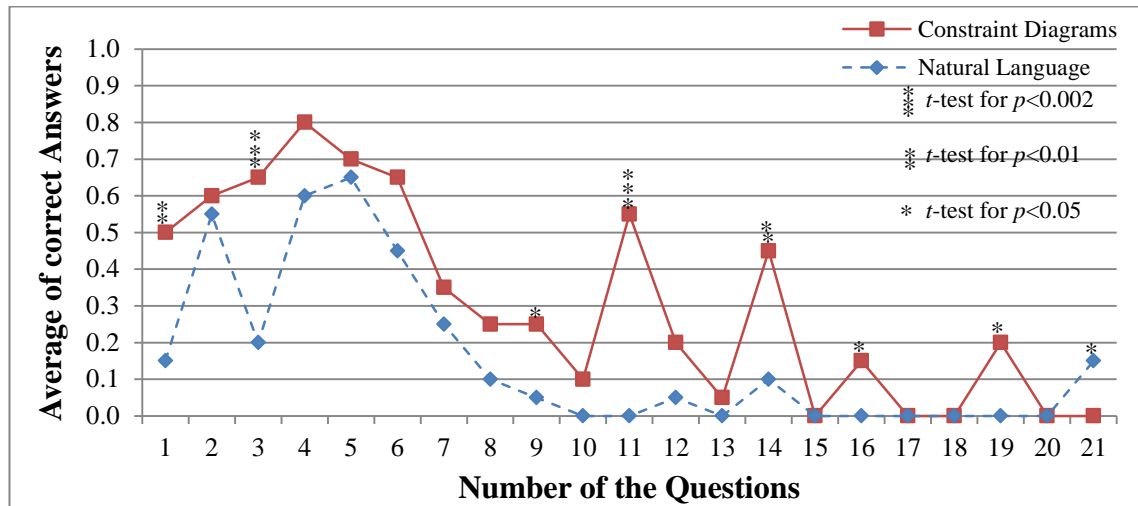


Figure 6.14 Graph of the average of correct answers for the two groups across the 21 questions

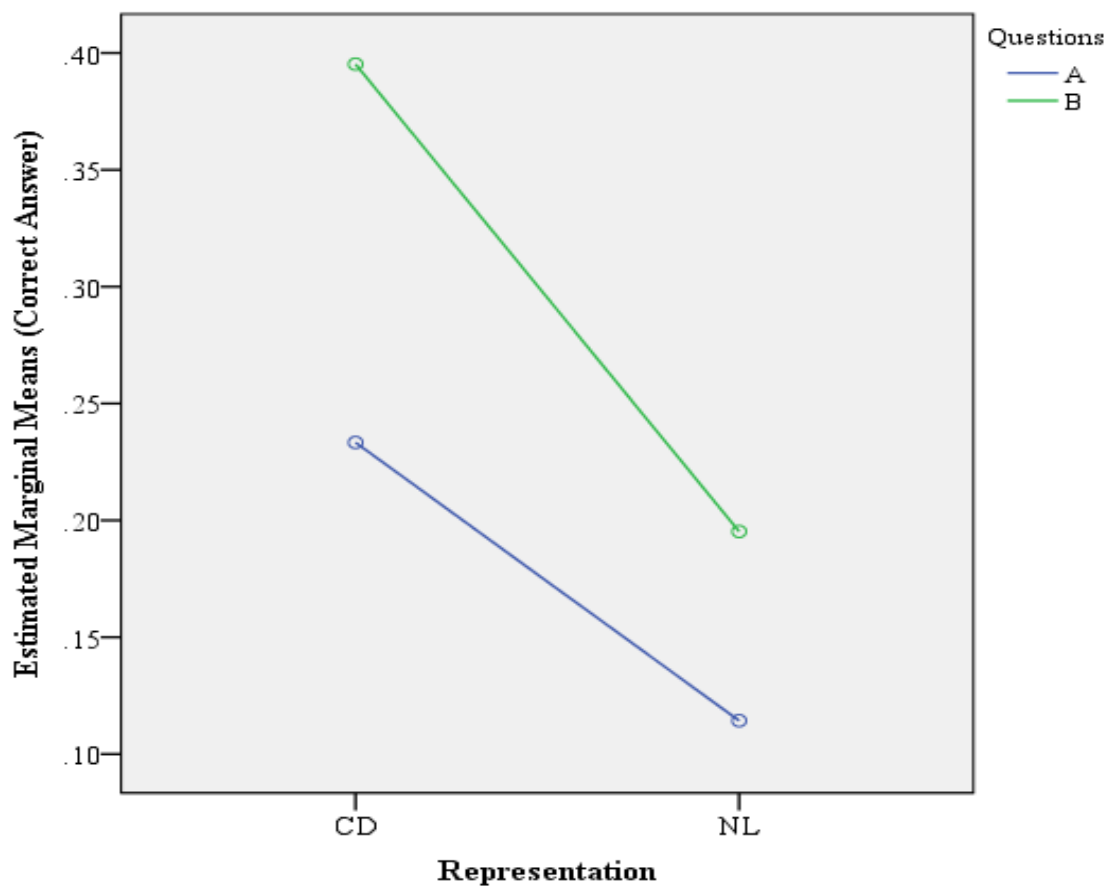


Figure 6.15 Graph of the interaction of CD and NL representations and correct answers of the two parts of the 21 questions

When using a mixed ANOVA design, there are significant results: the main within-subject effect of the two parts of the questions shows that $F(1,18)=60.803$ for $p<0.01$ and $\eta^2=0.772$, the main between-subject effect gives that $F(1,18)=12.389$ for $p<0.01$ and $\eta^2=0.408$, and the interaction effect is also significant: $F(1,18)=6.756$ for $p<0.01$ and $\eta^2=0.273$. The graph in Figure 6.11 shows that part A of the questions was difficult to answer for both groups. However, the difference between part A and part B was bigger with the CD group. We can look at the proportion of participants who got a particular question wrong; the assumption here is that the more participants who get a question wrong, the more difficult that question.

Since there are significant results, we will use t tests to find out these significances. There was a significant difference between the two groups in eight of the questions, according to t tests (one-tailed). The difference between the groups in Q1 was considerable: $t(18)=0.009$ ($p<0.01$); in Q3: $t(18)=0.0016$ ($p<0.002$); in Q11: $t(18)=0$ ($p<0.002$); in Q14: $t(18)=0.006$ ($p<0.01$); in Q9: $t(18)=0.040$ ($p<0.05$); in Q16: $t(18)=0.037$ ($p<0.05$); in Q19: $t(18)=0.018$ ($p<0.05$); and similarly for Q21, $t(18)=0.037$ ($p<0.05$). As shown in Figure 6.10, the CD group had more correct answers than the NL group (in 16 questions). In only one question, the NL group had more correct answers than the CD group. Overall, the CD group performed substantially better than the NL group.

By looking at the correct answers as concepts, there is a significant difference between the two groups in all concepts except concept 7. In concept 1 (Q1 to Q3): $t(18)=0.0008$ ($p<0.002$), concept 2 (Q4 to Q6): $t(18)=0.044$ ($p<0.05$), concept 3 (Q7 to Q9): $t(18)=0.022$ ($p<0.05$), concept 4 (Q10 to Q12): $t(18)=0$ ($p<0.002$), concept 5 (Q13 to Q15): $t(18)=0.007$ ($p<0.01$) and concept 6 (Q16 to Q18): $t(18)=0.040$ ($p<0.05$).

One way to consider the relative impact of the two representations compared to other factors on the difficulty of giving answers is to determine the strength of the correlation between the proportions of correct answers for each group. The greater the correlation the less likely that aspects specific to one or other representation is responsible for the level of performance. For the Pearson Product Moment Correlation, $r(19)=0.789$, which is significant at $p<0.002$. Since the two variables were strongly correlated, it is likely that none of the representations is responsible for the level of the performance,

As a result, the CD group had more correct answers than the NL group. Thus, the CD group performed substantially better than the NL group.

6.5.1.1. *Percentage of Correct Objects*

The correct-objects measurement was used to find out which notation would provide more correct objects that reflect participants' understanding of using that notation. If using CD notation to produce the specification will produce more correct used objects than using NL, then CD notation will be better than NL. We believe that CD notation will be worse than NL because it is a new notation and participants need to learn it and understand it by using the different objects in different places.

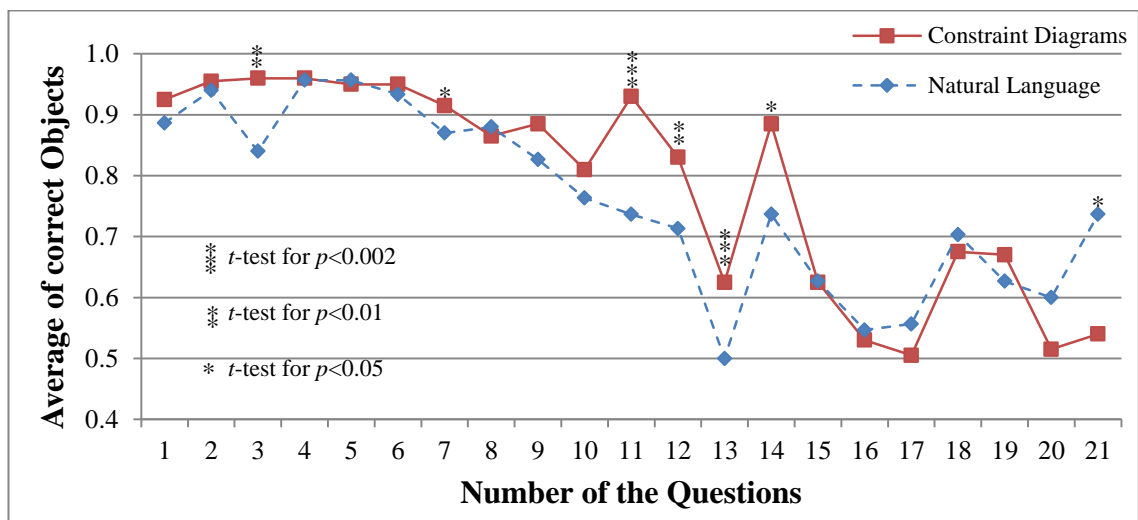


Figure 6.16 Graph of the average of correct objects for the two groups across the 21 questions

Figure 6.16 shows the mean number of correct objects for each question for the two groups. The overall mean number of correct objects by group is largely similar across the questions. There appears to be a general trend for the proportion of correct-objects to decrease from the earlier to the later questions. The mean (and SD) of the CD and NL groups over all of the questions were 0.786 (0.060) and 0.759 (0.045) respectively.

For both parts of the questions, part A and part B, the main within-subject effect is significant: $F(1,18)=61.484$ for $p<0.01$ and $\eta^2=0.774$. However, there are no significant results for both the main between-subject effect, $F(1,18)=1.336$ for $p<0.05$, and the interaction effect, $F(1,18)=0.879$ for $p<0.05$. Figure 6.17 shows that the difference between part A and part B is the same for both groups. However, the CD group have more correct objects than the NL group.

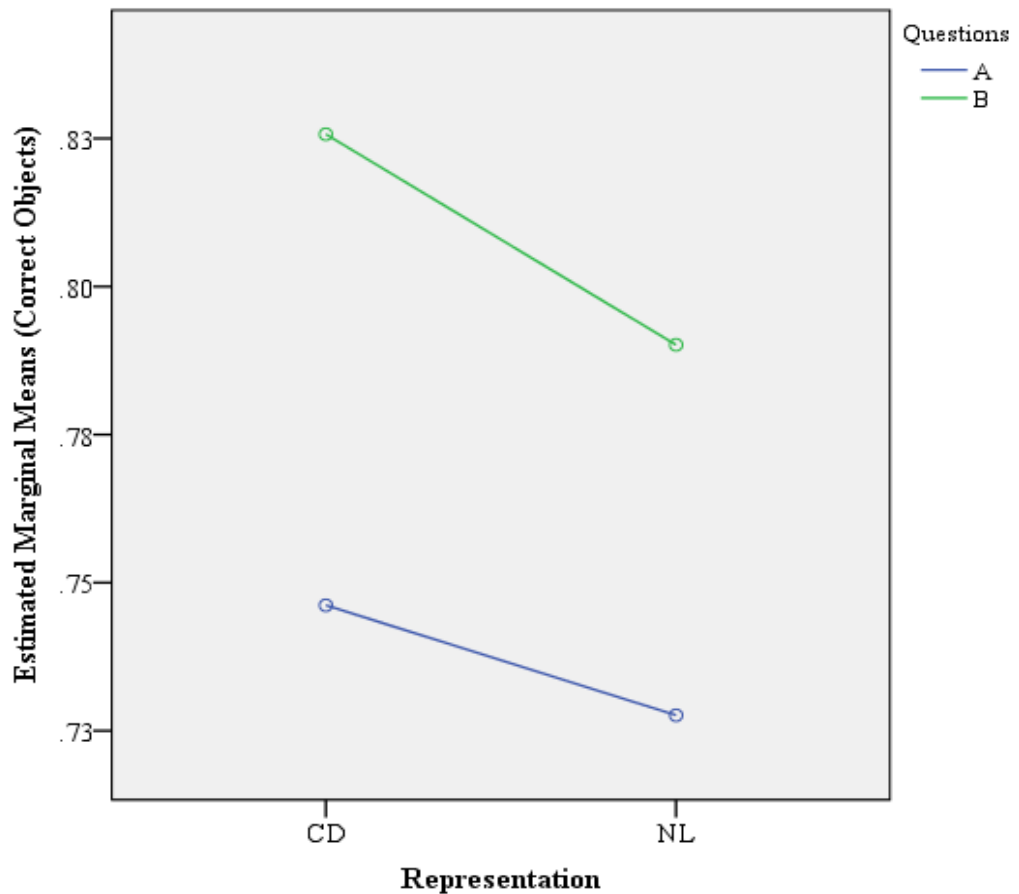


Figure 6.17 Graph of the interaction of CD and NL representations and correct objects of the two parts of the 21 questions

Since the main within-subject effect is significant, we will use *t* test to check for that significance. There is a significant difference between the two groups in seven of the questions, according to *t* tests (one-tailed). The difference between the groups in Q3 is considerable: $t(18)=0.005$ ($p<0.01$); in Q7, $t(18)=0.049$ ($p<0.05$); in Q11, $t(18)=0$ ($p<0.002$); in Q12, $t(18)=0.006$ ($p<0.01$); in Q13, $t(18)=0.001$ ($p<0.002$); in Q14, $t(18)=0.010$ ($p<0.05$) and similarly for Q21, $t(18) = 0.017$ ($p<0.05$). As shown in Figure 6.12, the CD group had more correct objects than NL (in 13 questions). In only eight questions, the NL group had more correct objects than the CD group. Overall, the CD group performed substantially better than the NL group.

By looking at the correct objects as concepts, there is a significant difference between the two groups in concept 1 (Q1 to Q3): $t(18)= 0.0022$ ($p<0.023$), concept 4 (Q10 to Q12): $t(18)= 0$ ($p<0.02$), concept 5 (Q13 to Q15): $t(18)=0.010$ ($p<0.01$) and concept 7 (Q19 to Q21): $t(18)=0.033$ ($p<0.05$).

By finding the Pearson Product Moment Correlation, $r(19) = 0.863$, is significant at $p < 0.002$, and thus, it is a strong correlation. So it is likely that none of the representations is responsible for the level of the performance.

Overall, the CD group performed substantially better than the NL group because they created more correct objects than the NL group.

6.5.1.2. *Percentage of Created Objects*

This measurement is used to find out which notations could cause a verbose answer. This could reflect participants' understanding of using that notation and their confidence in providing the answer. If using CD notation results in having the number of created objects equal to the number of the target, which is the correct objects, then this notation is better. However, if the number of created objects is higher than the target, then this notation is verbose. If the number of created objects is lower, then this language cannot specify the requirements. In general, if using CD notation to produce the specification will produce more or fewer objects than the target, then CD notation will be worse than NL, which is the hypothesis here since it is a new notation and participants need to explain their answers by being productive to reflect their learning and understanding of such new notation.

For the validity of the comparison, we found the number of objects created by participants for each question and divided it by the number of targeted correct objects for that question.

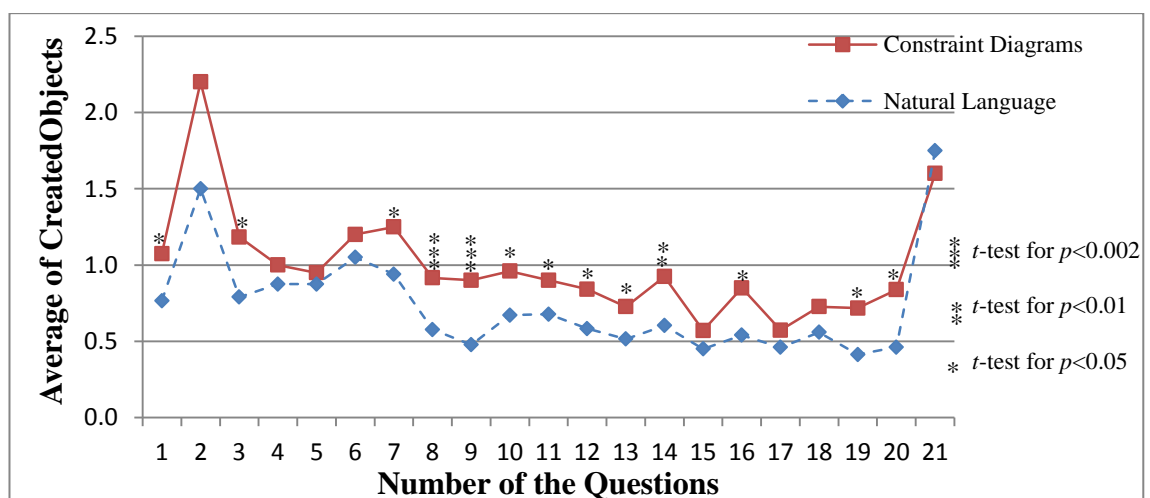


Figure 6.18 Graph of the average of created objects for the two groups across the 21 questions

Figure 6.18 shows the mean of the correct number of created objects to the target for each question for the two groups. The overall mean number of created objects by group is largely similar across the questions. There appears to be a general trend for the proportion of created objects to increase from the earlier to the later questions. The mean (and SD) of the CD and NL groups over all of the questions were 4.67 (3.35) and 4.67 (3.96) respectively.

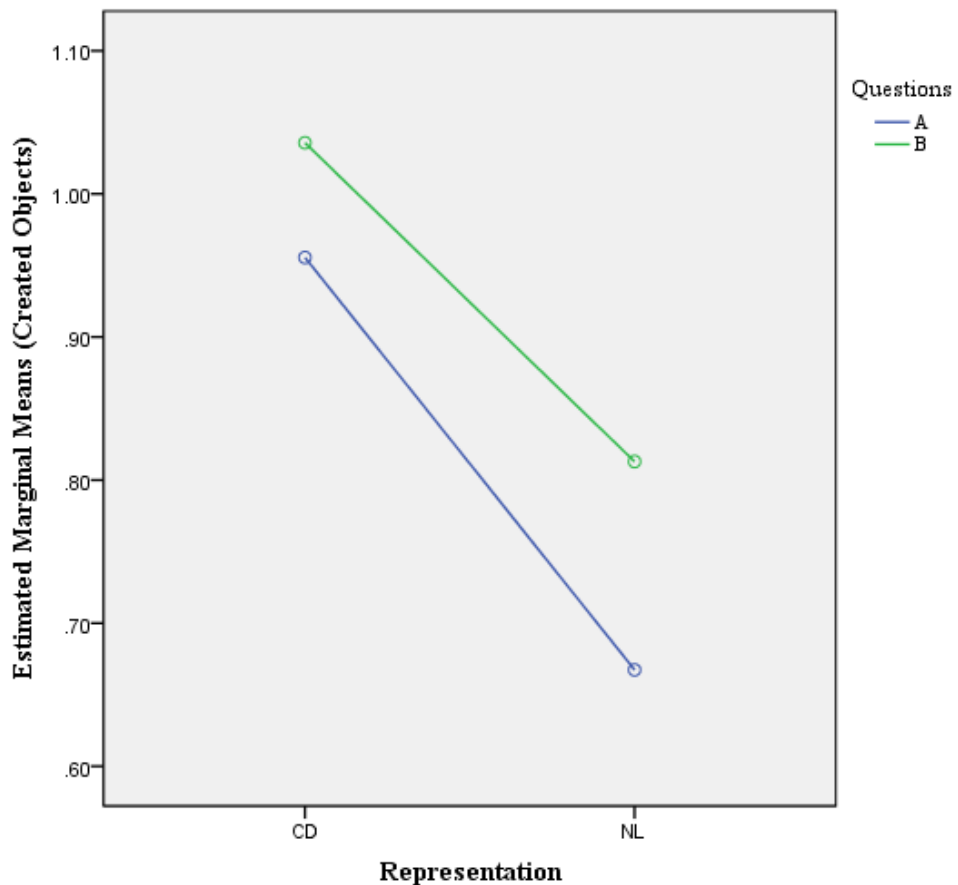


Figure 6.19 Graph of the interaction of CD and NL representations and created objects of the two parts of the 21 questions

Figure 6.19 shows that participants from both groups were more productive in part B questions. The CD group created more objects than the target compared with the NL group for the questions in part A which were informal questions. Although when specific information was required as in part B, the CD group also created more objects than the NL group, the difference between the two parts within the CD group was less than the difference between the two parts within the NL group. The interaction effect is not significant: $F(1,18)=1.57$ for $p<0.23$. However, the main between-group effect is

significant: $F(1,18)=6.31$ for $p<0.05$ and $\eta^2=0.260$. Moreover, the main within-subject effect is also significant: $F(1,18)=18.72$ for $p<0.01$ and $\eta^2=0.510$.

Since there are significant differences between groups, we will find the t -test results. There is a significant difference between the two groups in 13 of the questions, according to t -tests (one-tailed). The difference between the groups in Q8 is considerable: $t(18)=0.0015$ ($p<0.002$), in Q9: $t(18)=0$ ($p<0.002$), in Q14: $t(18)=0.005$ ($p<0.01$); in Q1: $t(18)=0.019$ ($p<0.05$); in Q3: $t(18)=0.017$ ($p<0.05$); in Q7: $t(18)=0.020$ ($p<0.05$); in Q10: $t(18)=0.020$ ($p<0.05$); in Q11: $t(18)=0.042$ ($p<0.05$); in Q12: $t(18)=0.015$ ($p<0.05$); in Q13: $t(18)=0.047$ ($p<0.05$); in Q16: $t(18)=0.019$ ($p<0.05$); in Q19: $t(18)=0.026$ ($p<0.05$); and similarly for Q20: $t(18)=0.046$ ($p<0.05$). As shown in Figure 6.18, in one question, Q4, the CD group had a number of created objects equal to the target created objects. Moreover, in one question, Q2, both groups had more created objects than the target. There were fewer created objects than the target in 19 questions for the CD group and in 20 questions for the NL group. As a result, the CD notation is not causing any need to provide a verbose explanation. Thus, the CD group performed substantially better than the NL group.

To check if a specific concept is causing a verbose answer, we will find by looking at the created objects as concepts that there is a significant difference between the two groups in concept 3 (Q7 to Q9): $t(18)=0$ ($p<0.002$); concept 4 (Q10 to Q12): $t(18)=0$ ($p<0.002$); concept 5 (Q13 to Q15): $t(18)=0.001$ ($p<0.002$); concept 6 (Q16 to Q18): $t(18)=0.003$ ($p<0.01$); and concept 1 (Q1 to Q3): $t(18)=0.036$ ($p<0.05$).

There is a strong correlation between the two representations on the created objects measurement. By using the Pearson Product Moment Correlation, we found that $r(19)=0.891$, which is significant at $p<0.002$.

As a result, CD notation is not causing any need to provide a verbose explanation. Thus, the CD group performed substantially better than the NL group.

6.5.1.3. *Percentage of Configuration*

Since the number of created and correct objects is calculated automatically, we need to make sure that this number reflects the target. For example, if there is a set called *Patient* and its subset is called *Alive*, then the number of targeted objects is two. However, these two objects could be drawn disjointed which is inaccurate configuration

as in Figure 6.20, drawn above each other with the same size and xy points which visually seem to be one object, which is also inaccurate configuration as in Figure 6.21, or drawn as *Patient* being a subset and *Alive* as the super set which also an inaccurate configuration as in Figure 6.22. Thus, this measurement is used to manually visualize the answer and to find out how these objects are really used. This could reflect participants' understanding of using that notation. If using CD notation results in having more accurate configurations than using NL notation, then CD notation is better, and vice versa.

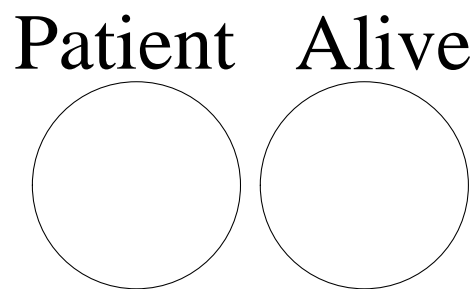


Figure 6.20 Two disjoint sets

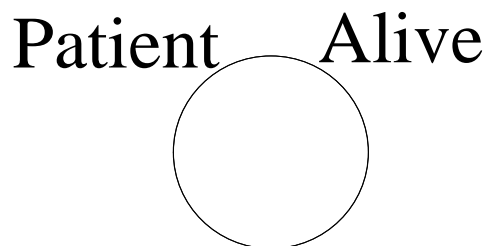


Figure 6.21 Two sets with the same size and xy points

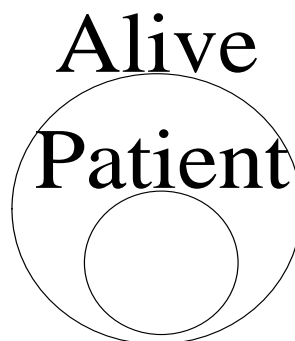


Figure 6.22 Two related sets

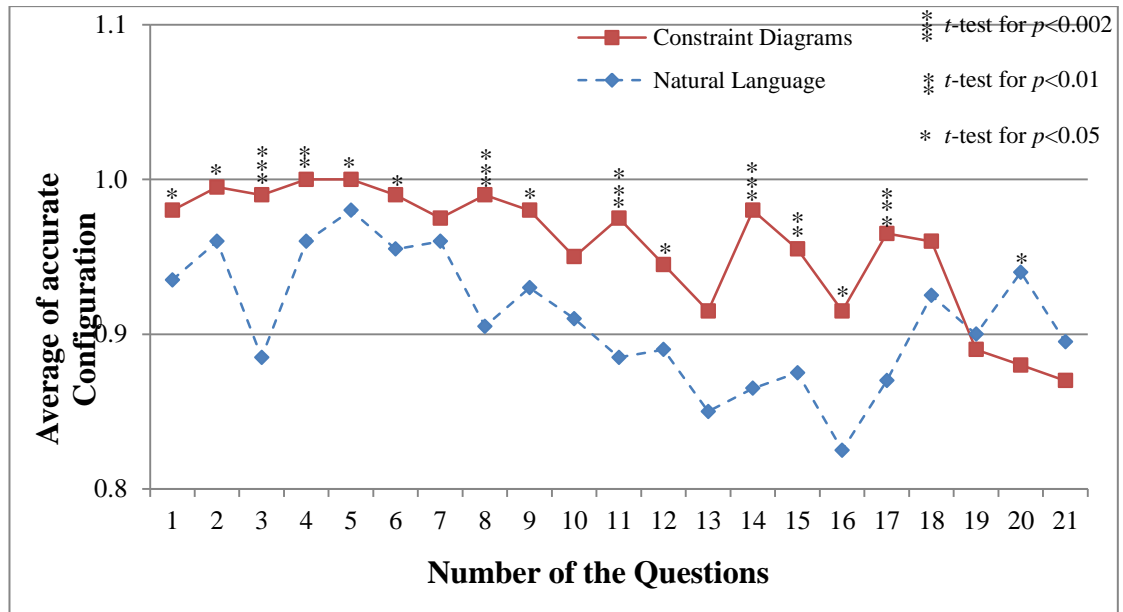


Figure 6.23 Graph of the average of configuration for the two groups across the 21 questions

Figure 6.23 shows the mean number of accurate configuration for each question for the two groups. The overall mean number of configuration by group is largely similar across the questions. There appears to be a general trend for the proportion of configuration to increase from the earlier to the later questions. The mean (and SD) of the CD and NL groups over all of the questions are 0.957 (0.074) and 0.909 (0.109) respectively.

There is no significant interaction effect for the two parts for the two groups: $F(1,18)=2.649$ for $p<0.05$ and also, there is no significant main within-subject effect of the two parts: $F(1,18)=2.278$ for $p<0.05$. However, the main between-subject effect for the two groups shows a significant result: $F(1,18)=8.086$ for $p<0.01$ and $\eta^2=0.310$. Figure 6.124 shows that the difference of the configuration in the CD group is very small.

Since the main between-subject effect is significant, t tests are conducted. There is a significant difference between the two groups in 15 of the questions, according to t tests (one-tailed). For $p<0.002$, the difference between the groups in Q3 is considerable: $t(18)=0$; in Q8: $t(18)=0$; in Q11: $t(18)=0.001$; in Q14: $t(18)=0.0017$; and similarly for Q17, $t(18)=0$. For $p<0.01$, the difference between the groups in Q4: $t(18)=0.006$, and in Q15: $t(18)=0.007$. For $p<0.05$; the difference between the groups in Q1 is considerable: $t(18)=0.012$; in Q2: $t(18)=0.018$; in Q5: $t(18)=0.048$; in Q6: $t(18)=0.032$;

in Q9: $t(18)=0.012$; in Q12: $t(18)= 0.026$; in Q16: $t(18)= 0.030$; and similarly for Q20, $t(18) = 0.023$. As shown in Figure 6.16, the CD group had more accurate configuration than NL (in 18 questions). In only three questions, the NL group had more accurate configuration than the CD group. Overall, the CD group performed substantially better than the NL group.

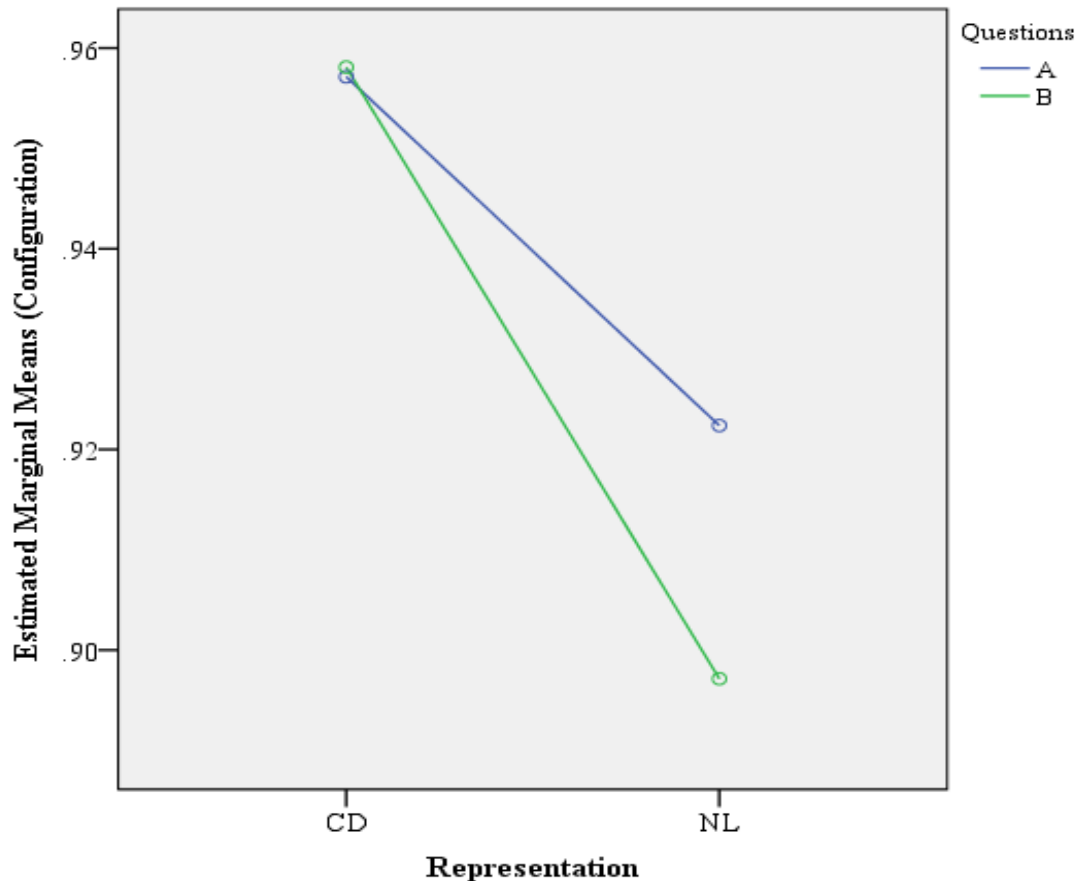


Figure 6.24 Graph of the interaction of CD and NL representations and configuration of the two parts of the 21 questions

By looking at the concepts configurations, there is a significant difference between the two groups in all concepts except concept 7. For $p<0.002$, concepts 1 to 6 have $t(18)=0$.

There is a strong correlation between the two representations on the configuration measurement at $p<0.05$. By using the Pearson Product Moment Correlation, we found that $r(19)= 0.414$, which, however, is not significant at $p<0.002$.

As a result, the CD group had more accurate configuration than the NL group and thus, they performed substantially better than the NL group.

6.5.2. Percentage of Steps

This measurement is used to find out which notation would require more steps to produce an answer. This could reflect participants' confidence and understanding of using that notation. If using CD notation results in having a number of steps equal to the number of the targeted steps, then this notation is better. However, if the number of steps is higher or lower than the target, then this notation would not specify the requirements sufficiently. In general, compared to the NL group, if using CD notation to produce the specification would need more or fewer steps than the target, then CD notation would be worse than NL.

For the validity of the comparison, we found the number of steps by participant for each question and divided it by the number of targeted steps for that question. For NL, the targets steps are equal to the number of correct objects plus one, for pressing the *Next* button. On the other hand, for CD, the targeted steps, if there is only one object, is equal to the number of correct objects plus one, for pressing the *Next* button, but if there is more than one object, then the number of the targeted steps equals three times the number of correct objects and one more step for pressing the *Next* button. The reason behind tripling the steps for CD is that each object needs to be created then sized at least once and then moved as well.

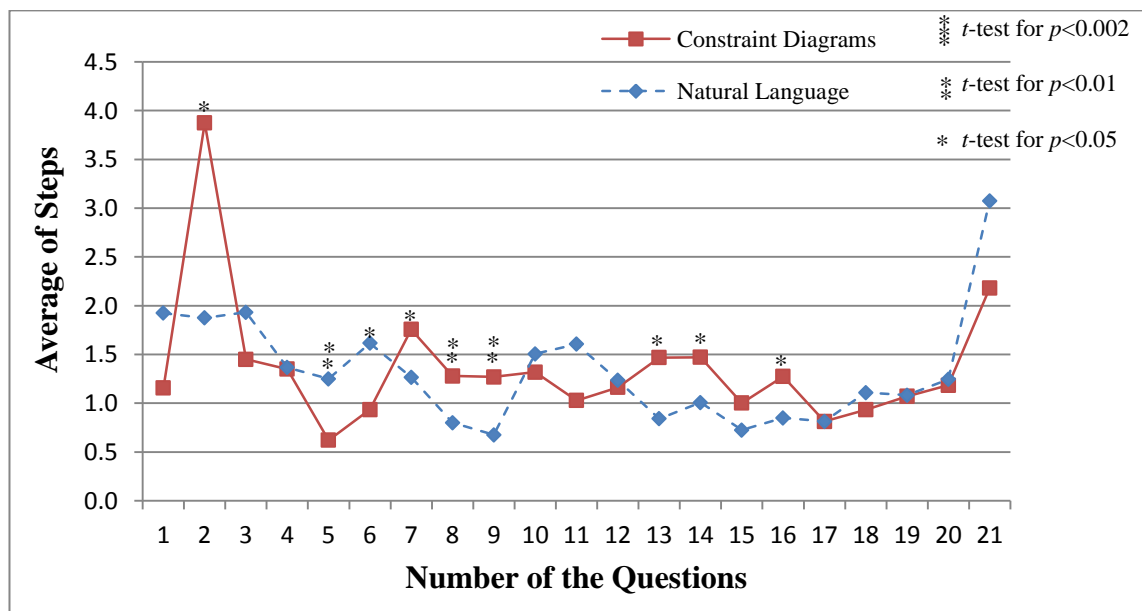


Figure 6.25 Graph of the average of steps for the two groups across the 21 questions

Figure 7.25 shows the mean number of steps for each question for the two groups. There appears to be a general trend for the percentage of steps to increase from the earlier to the later questions. The mean (and SD) of the CD and NL groups over all of the questions are 1.36 (1.30) and 1.32 (1.26) respectively.

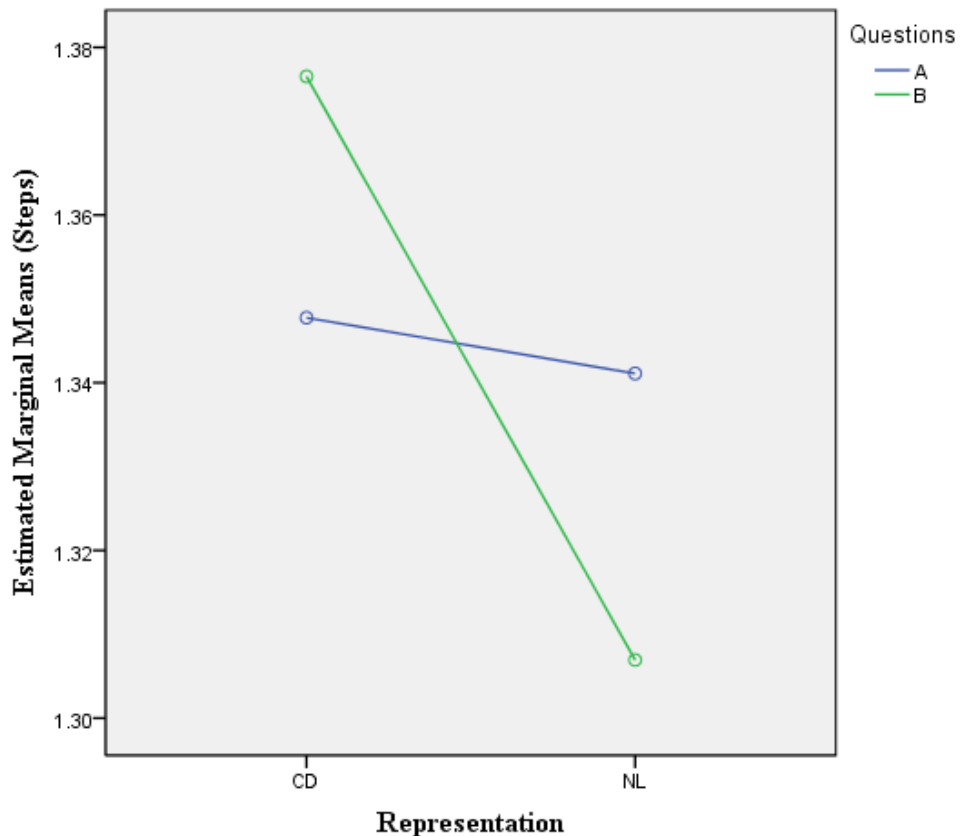


Figure 6.26 Graph of the interaction of CD and NL representations and steps of the two parts of the 21 questions

Figure 6.26 shows that there are statistically no significant results. The interaction effect for the two parts for the two groups is not significant given that $F(1,18)=0.214$ for $p<0.65$. Moreover, there is no significant main between-subject effect: $F(1,18)=0.028$ for $p<0.87$. Furthermore, the main within-subject effect is not significant: $F(1,18)=0.002$ for $p<0.97$. However, in part B questions, the CD group had more steps than part B while the NL group had fewer steps for part B than for part A. It may be the case that CD is a diagrammatic notation and needs to be sized and moved to provide a good diagram. As shown in the plot, part B for CD group is worse than the NL group as well as worse than part A for both, but due to the representation there is no difference because the significance results are more than 0.05. According to the significance results, there is no significant difference between both parts and no significant difference between both representations. Also, there are no interactions between representations and parts.

Despite the fact that ANOVA test showed that there are no significant results, we will show t tests as well. There is a significant difference between the two groups in nine of the questions, according to t tests (one-tailed). For $p < 0.01$, the difference between the groups in Q5 is considerable: $t(18) = 0.002$, in Q8: $t(18) = 0.005$; and in Q9: $t(18) = 0$. For $p < 0.05$, the difference between the groups in Q2 is considerable: $t(18) = 0.039$; in Q6: $t(18) = 0.050$; Q7: $t(18) = 0.011$; Q13: $t(18) = 0.012$; Q14: $t(18) = 0.037$; and similarly for Q16, $t(18) = 0.037$. As shown in Figure 6.25, the CD group had a greater number of steps than the targeted steps (in only one question, Q2). Overall, the CD group performed as well as the NL group.

By looking at the steps for concepts, there is significant difference between the two groups in three concepts. For $p < 0.01$, concept 2: $t(18) = 0.008$; and for $p < 0.002$ concept 3: $t(18) = 0$ and for concept 5: $t(18) = 0.001$.

There is a strong correlation between the two representations on the steps measurement at $p < 0.05$. By using the Pearson Product Moment Correlation, we found that $r(19) = 0.444$, which, however, is not significant at $p < 0.002$.

By examining the kind of available and used actions (steps) we found that actions such as Resize, Bring to front and Send to back are not used in the NL version. However, this group used steps such as Cancel Operation and Return to Example, approximately twice as often as the CD group, which, in the other hand, used steps such as Move, Show popup menu, Resize, Bring to front, and Send to back. Steps such as Add, Move by outside, Delete, and Change label were similar in both groups.

As a result, the CD group and the NL group had no significant differences in performing steps, and thus the CD notation is as good as the formal NL notation.

6.5.3. Percentage of the Number of Returns to the Examples

This measurement is used to find out which notation required more understanding by returning to the examples to learn the concepts before being able to produce an answer. This could reflect participants' confidence and understanding of what they learnt using that notation. If using CD notation results in fewer returns than using NL notation, then this notation (CD) is better because they could understand quicker than with the other notation. We believe that CD notation will be worse than NL because it is a new notation and participants need to learn it and understand it.

It was predicted that both groups would return to the examples more often, especially the CD group. Figure 6.27 shows the mean number of returns for each example for the two groups. There appears to be a general trend for the percentage of returns to increase from the earlier to the later questions. The mean (and SD) of the CD and NL groups over all of the questions are 5.885 (10.527) and 9.957 (16.737) respectively.

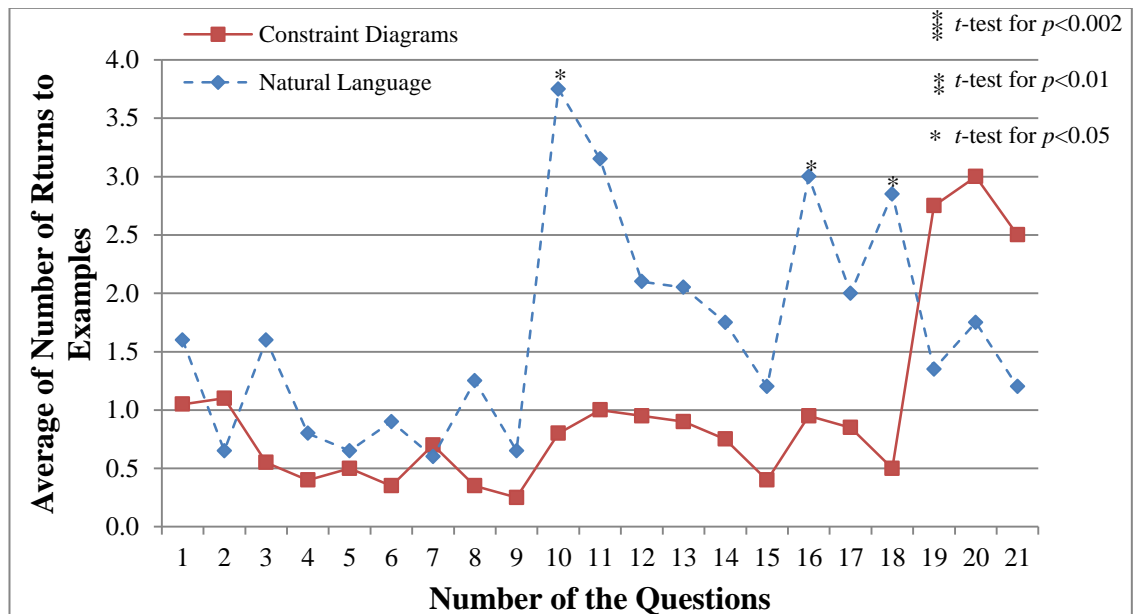


Figure 6.27 Graph of the average of returns to examples for the two groups across the 21 questions

As shown in Figure 6.28, the interaction effect, the main between-subject effect and the main within-subject effect are not significant: with $F(1,18)= 1.089$ for $p<0.05$, $F(1,18)=0.0673$ for $p<0.05$, and $F(1,18)= 3.277$ for $p<0.05$. As shown in the plot, questions part B for CD group is worse than the NL group as well as worse than the first half for both, but due to the representation there is no difference because the significance results are more than 0.05. According to the significance results, there is no significant difference between both halves and no significant difference between both representations. Also, there are no interactions between representations and the two parts.

There was significant difference between the two groups in three of the questions, according to t tests (one-tailed). For $p<0.05$, the difference between the groups in Q10 was considerable: $t(18)=0.013$; in Q16, $t(18)=0.045$; and similarly for Q18, $t(18) = 0.024$. As shown in Figure 6.27, the NL group had more returns to examples than the CD group (in only 16 questions). In only five questions, the CD group had more returns than the NL group. Overall, the NL group returned more to examples and, and thus the CD group performed better than the NL group.

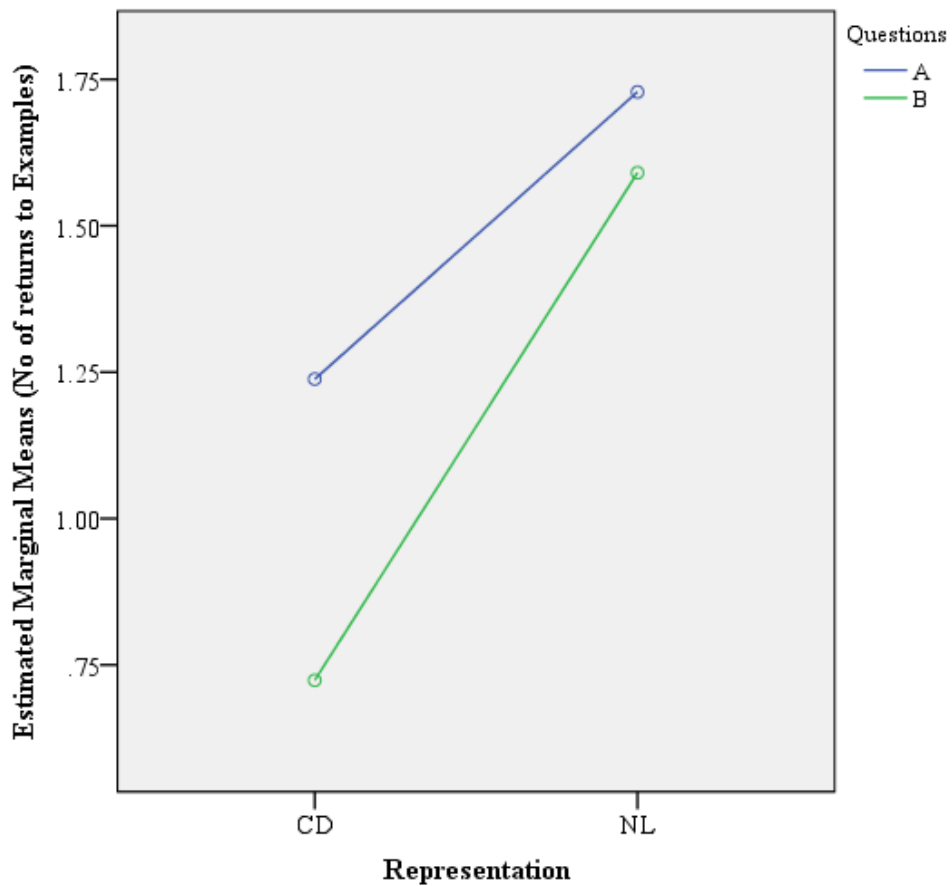


Figure 6.28 Graph of the interaction of CD and NL representations and steps of the two parts of the 21 questions

By looking at the number of returns for the concepts represented by the examples, we found that there is no significant difference between the two groups in all concepts.

By using the Pearson Product Moment Correlation, we found that $r(19) = 0.055$, which is not significant at $p < 0.05$ and thus there is no correlation between the two representations on the number of returns to the examples measurement at $p < 0.002$.

As a result, the CD group had fewer returns to examples than the NL group, and thus they performed better than the NL group.

6.5.4. Time Spent on each Example

This measurement is used to find out which notation requires less time on studying the concepts, which will reflect the ease of understanding the use of that notation. If using CD notation will require less time than using NL, then CD notation will be better than NL. We believe that CD notation will be worse than NL because it is a new notation and participants need to spend more time to learn it and understand it.

Time spent on each example is calculated by finding the time spent on each new example that is introduced to participants for the first time in this training (in section: 6.5.4.1), and the time spent on each returned-to example (in section: 6.5.4.2).

Figure 6.22 shows the mean amount of time spent on studying each example, for each new (also discussed in section 6.5.4.1) and each returned (also discussed in section 6.5.4.2) example for the two groups. The overall mean amount of time by each group is largely different across the examples. Unlike for the NL group, there appears to be a general trend for the amount of time spent on examples by the CD group to increase from the earlier to the later examples. The mean (and SD) of the CD and NL groups over all of the examples are 129.84 (96.80) and 133.68 (97.10) respectively.

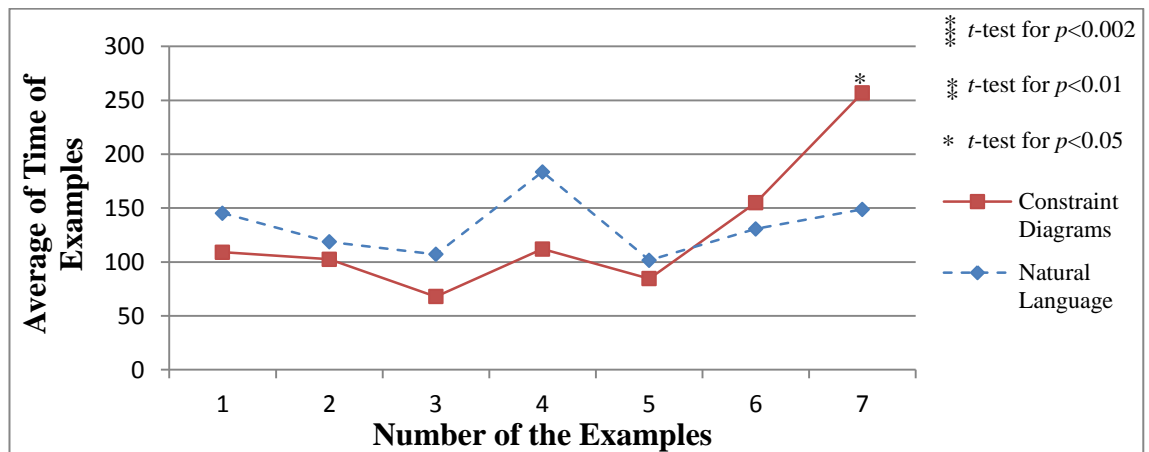


Figure 6.29 Graph of the average of time spent on each example for the two groups across the 7 examples

There is no significant difference between the two groups in all but one of the examples, according to t tests (one-tailed). The difference between the groups in E7 is $t(5)=0.047$ ($p < 0.05$). The CD group achieved a time that was less than that of the NL group in all examples except examples 6 and 7. As shown in Figure 6.22, the CD group spent less time on returned-to examples than the NL group (in five examples). Overall, the CD group spent less time on the examples, and thus the CD group performed better than the NL group.

There is no significant correlation between the two representations on the Time-Spent-on-Examples measurement. For the Pearson Product Moment Correlation: $r(5)=.418$, which is not significant at $p < 0.05$.

As a result, the CD group spent less time on studying the examples than the NL group and thus they performed better than the NL group in learning the notation.

6.5.4.1. Time Spent on each new Example

This measurement is used to find the time spent on each new example that was introduced for the first time. If using CD notation will require less time than using NL, then CD notation will be better than NL. We believe that CD notation will be worse than NL because it is a new notation and participants need to spend more time to learn it and understand it.

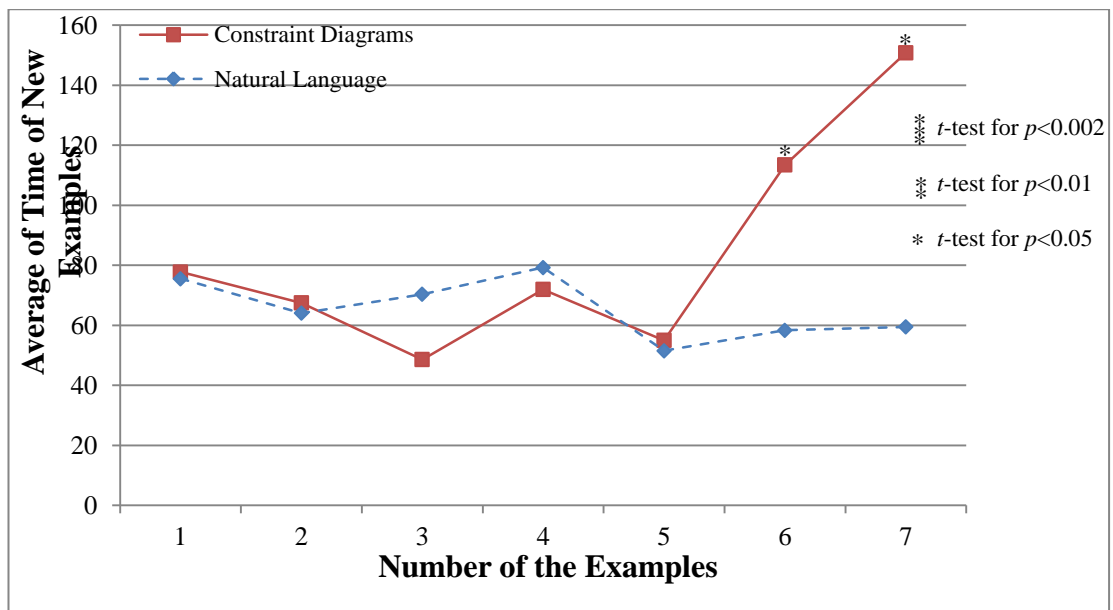


Figure 6.30 Graph of the average of time spent on each new example for the two groups across the 7 examples

Each participant was directed to a new example to be studied for the first time independently of any question. Figure 6.30 shows the mean amount of time for each example for the two groups. The overall mean amount of time spent by each group is largely different across the examples. Unlike the NL group, there appears to be a general trend for the CD group for the amount of time to increase from the earlier to the later examples. The mean (and SD) of the CD and NL groups over all of the examples are 83.57 (61.98) and 65.51 (50.10) respectively.

There is no significant difference between the two groups in all but two of the examples, according to t tests (one-tailed). For $p < 0.05$, the difference between the groups in E6 is significant, $t(18)=0.024$, and similarly for E7, $t(18)=0.017$. The CD group achieved a

time that was less than that of the NL group in two examples: E3 and E4. Overall, the CD group performed substantially worse than the NL group.

For the Pearson Product Moment Correlation was $r(5)=-0.269$, which is not significant at $p<0.05$. To clarify, the same examples were of comparable difficulty for both groups. This suggests again that the performance of the two groups was not substantially different.

As a result, the CD group spent more time on each new example introduced for the first time than the NL group, and thus they performed substantially worse than the NL group.

6.5.4.2. Time Spent on each Returned Example

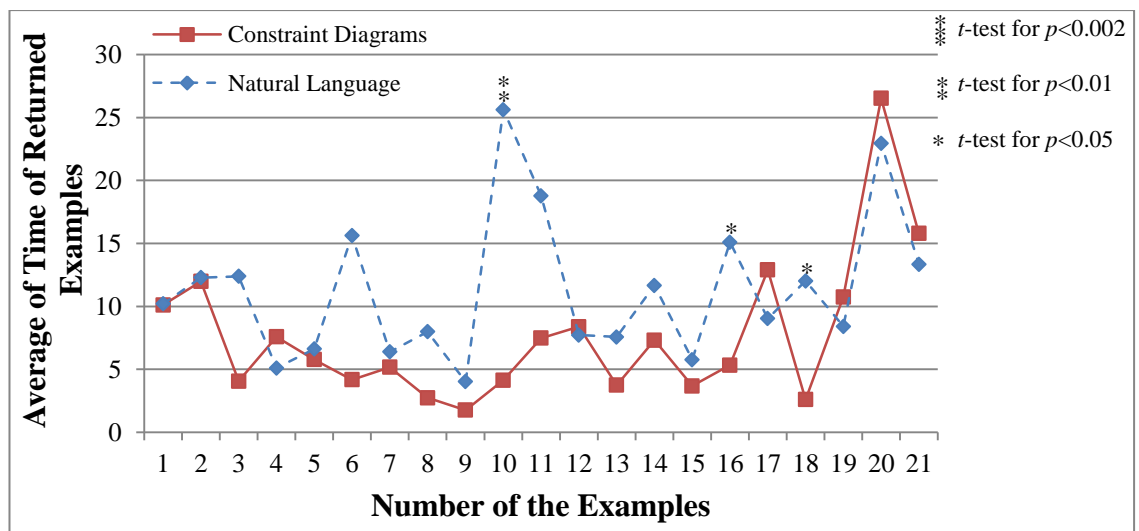


Figure 6.31 Graph of the average of time spent on each returned example for the two groups across the 21 questions

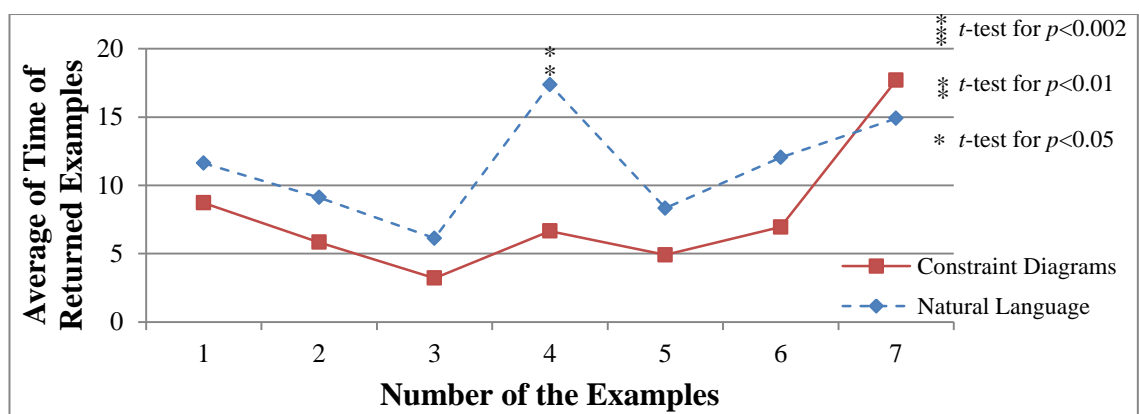


Figure 6.32 Graph of the average of time spent on each returned example for the two groups across the 7 examples

This measurement is used to find the time spent on each returned-to example which will reflect the ease of understanding the use of that notation. If using CD notation will

require less time than using NL, then CD notation will be better than NL. We believe that CD notation will be worse than NL because it is a new notation and participants need to spend more time to learn it and understand it.

Figure 6.32 shows the mean time spent on each return to an example for the two groups. The overall mean of the time spent on returns by each group is largely different across the examples. There appears to be a general trend for the time spent on returns to increase from the earlier to the later examples. The mean (and SD) of the CD and NL groups over all of the examples are 46.27 (66.77) and 68.17 (77.90) respectively.

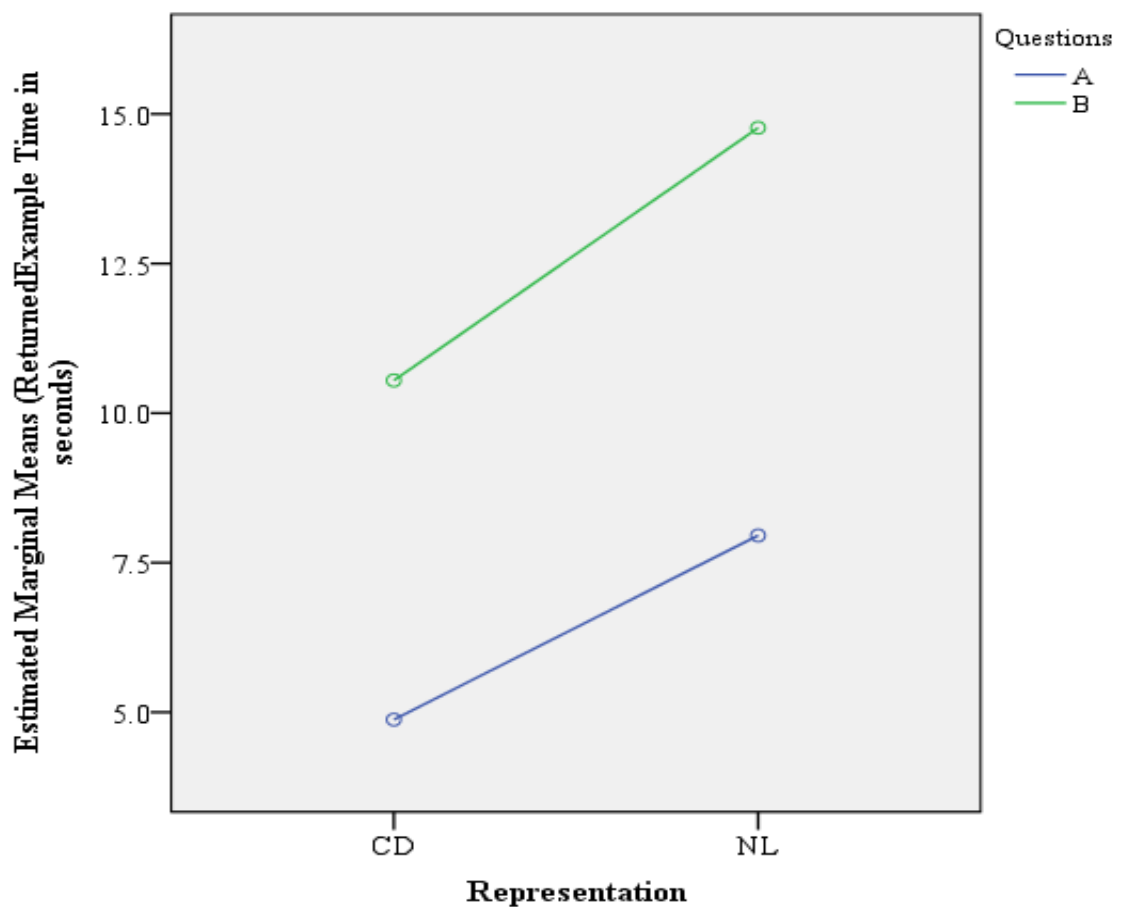


Figure 6.33 Graph of the interaction of CD and NL representations and the returned examples time of the two parts of the 21 questions

Figure 6.33 shows that for the two groups and for the two parts, the interaction effect and the between-subject effect are not significant, with $F(1,18)=0.998$ for $p<0.05$ and $F(1,18)=0.113$, respectively. However, there is a significant main within-subject effect with $F(1,18)=13.503$ for $p<0.01$ and $\eta^2=0.429$. Figure 6.33 shows that part B is worse than the part A for both representations. For CD group, they take less training time in both parts compared with NL group. Both CD and NL groups take more time in part B

compared to part A. However, in general, there are no differences in neither both representations and both parts.

There is significant difference between the two groups in three of the questions, according to t tests (one-tailed). For $p < 0.01$, the difference between the groups in Q10 is considerable: $t(18) = 0.005$ and for $p < 0.05$; the difference in Q16 is $t(18) = 0.027$; and similarly for Q18, $t(18) = 0.048$. As shown in Figure 6.31, the NL group spent more time on returned-to examples than the CD group (in 13 questions). In five questions, the CD group spent more time than the NL group. Overall, the NL group spent more time on returning to examples, and thus the CD group performed better than the NL group.

By looking at Figure 6.32, we find that according to the time spent on returned-to examples that represent concepts, there is no significant difference between the two groups in all concepts except in E4: $t(18) = 0.008$ (for $p < 0.01$).

There is a strong correlation between the two representations in the time spent on each returned-to example measurement at $p < 0.05$. By using the Pearson Product Moment Correlation, we found that $r(19) = 0.379$, which, however, is not significant at $p < 0.002$.

As a result, the CD group spent less time on returned-to examples than the NL group, and thus they performed substantially better than the NL group.

6.5.5. Time Spent on each Initial-Thinking

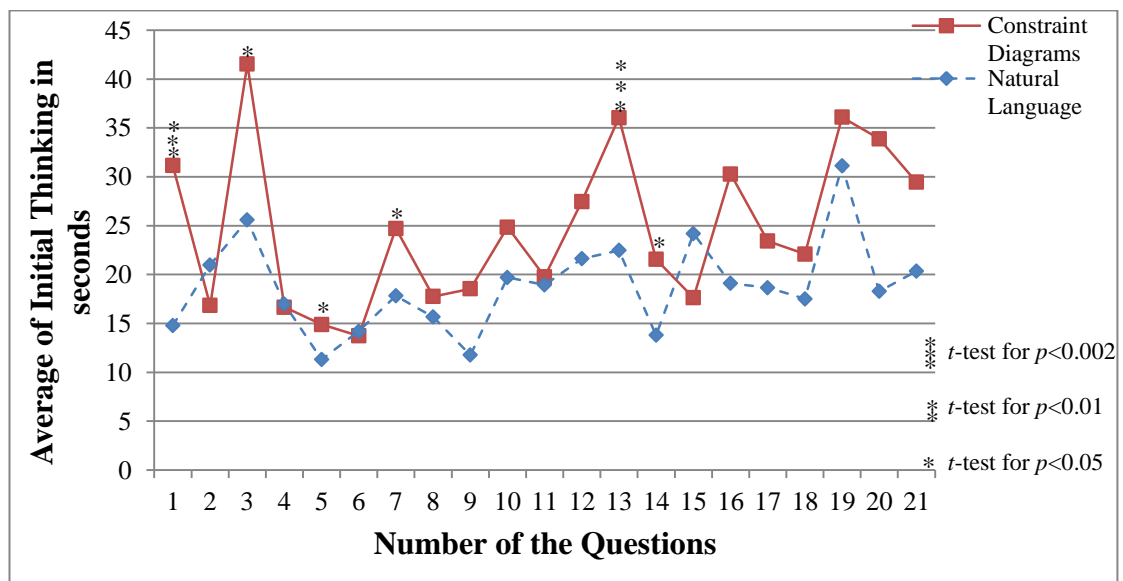


Figure 6.34 Graph of the average of time spent on each initial thinking for the two groups across the 21 questions

This measurement is used to find out which notation required more time in studying the concepts before the first action was performed by the participant after reading and understanding the question. This will reflect the ease of understanding of how to use that notation. If using CD notation will require less time than using NL, then CD notation will be better than NL. We believe that CD notation will be worse than NL because it is a new notation and participants need to spend more time to learn it and understand it. We will not compare the time taken by both groups to answer the questions because the editor used for every notation has its own tools and this requires different actions, and thus the time spent on performing the answer is affected by different factors.

It was predicted that participants would need a lot of initial thinking time to answer questions, especially the CD group. Figure 6.34 shows the mean amount of initial time for each question for the two groups. There appears to be a general trend for the initial thinking time to decrease from the earlier to the later questions. The mean (and SD) of the CD and NL groups over all of the questions are 24.69 (20.52) and 18.80 (14.67) respectively.

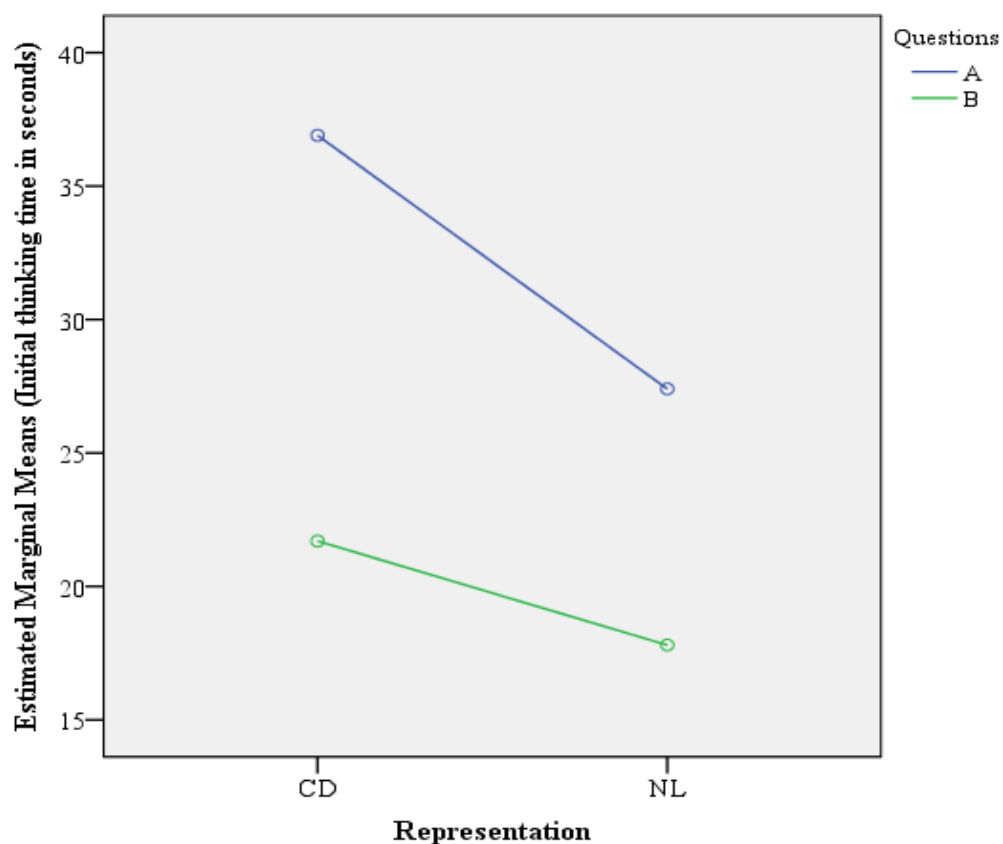


Figure 6.35 Graph of the interaction of CD and NL representations and initial thinking time of the two parts of the 21 questions

Figure 6.35 shows that there is a significant main within-subject effect with $F(1,18)=64.817$ for $p<0.01$ and $\eta^2=0.783$. Moreover, the main between-subject effect is significant with $F(1,18)=4.833$ for $p<0.05$ and $\eta^2=0.212$. However, there is no significant interaction effect: $F(1,18)=3.305$ for $p<0.05$. Figure 6.35 shows that part A is worse than the part B for both representations. For NL group, they take less training time in both parts compared with the CD group. Both CD and NL groups take more time in part A compared to part B. However, in general, there are no differences in neither both representations and both parts.

Since the between-subject effect is significant, t tests are performed. There is no significant difference between the two groups in all but 6 of the questions, according to t tests (one-tailed). The difference between the groups in Q1 is considerable: $t(18)=0.001$ ($p<0.002$), in Q3: $t(18)=0.032$ ($p<0.05$), in Q5: $t(18)=0.033$ ($p<0.05$); in Q7: $t(18)=0.047$ ($p<0.05$), in Q13: $t(18)=0.017$ ($p<0.002$); and in Q14: $t(18)=0.009$ ($p<0.05$). As shown in Figure 6.34, the NL group needed less initial thinking time to answer the questions (in 17 questions). In four questions the CD group needed less initial thinking time than the NL group. Overall, the NL group performed substantially better than the CD group.

By looking at the initial thinking time measurement for the concepts used, there is a significant difference between the two groups in four concepts: E1: $t(18)=0.013$ ($p<0.05$); E3: $t(18)=0.030$ ($p<0.05$); E6: $t(18)=0.017$ ($p<0.05$); and E7: $t(18)=0.002$ ($p<0.01$).

By using the Pearson Product Moment Correlation, we found that there is a strong correlation between the two representations on the initial thinking time measurement: $r(19)=0.379$, which is significant at $p<0.002$.

As a result, the CD group needed more initial thinking time than the NL group, and thus they performed substantially worse than the NL group.

6.5.6. Relations between different measures

In this subsection, we explore any evidence of similarities between the two groups and between variables by using the Pearson Product Moment Correlation.

The Bonferroni correction is used here as well by dividing the conventional significance level, which is 0.05, by the number of tests, which is 20 – 10 measurements for each group, to find the adjusted significance level, which is ($p < 0.0025$).

Table 6.1 shows the Pearson Product Moment Correlation value for the CD group on all the measures: steps, number of returns to examples, initial thinking time, created objects, correct objects, configuration, correct answers, example time, new-example time, and returned-to-examples time. There is a tendency for participants who often returned to examples to spend more time on these returned examples, which is statistically significant for $p < 0.002$; and also more returns to examples and more steps to be performed for $p < 0.05$ (Table 6.1). Another tendency is a greater number of created objects, and more steps to be performed, which is significant for $p < 0.01$; the more accurate configuration as well as more example time, which all are significant at $p < 0.05$. Moreover, initial thinking time tends to mean more example time, which is significant at $p < 0.01$, and more new-example time, which is significant for $p < 0.05$. Further, more new-example time means more example time to record, which is significant at ($p < 0.05$). However, there are no significant relations between the other measurements.

Table 6.1 The Pearson Product Moment Correlation between different measures for the CD group

Pearson Product Moment Correlation	Steps	No. Returns	Initial Thinking	Returned-to Example Time	Created Objects	Correct Objects	Configuration	Correct Answers	Example Time
No. Returns	.490								
Initial Thinking	.413	.076							
Returned-to Example Time	.603*	.888***	.213						
Created Objects	.787**	.383	.399	.456					
Correct Objects	-.073	.216	-.264	.079	.205				
Accurate Configuration	.545	.327	.212	.304	.652*	.474			
Correct Answers	.077	.413	.117	.228	-.039	.589*	.295		
Example Time	.415	.436	.737**	.547	.558*	.316	.530	.433	
New-Example Time	-.054	-.283	.662*	-.253	.241	.295	.344	.299	.671*

Legend: Level of significant for one-tailed, *** $p < 0.0025$, ** $p < 0.01$ level, * $p < 0.05$ level for (df=8).

Table 6.2 The Pearson Product Moment Correlation between different measures for the NL group

Pearson Product Moment Correlation	Steps	No. Returns	Initial Thinking	Returned-to Example Time	Created Objects	Correct Objects	Configuration	Correct Answers	Example Time
No. Returns	.840***								
Initial Thinking	.224	.496							
Returned-to Example Time	.784**	.945***	.599*						
Created Objects	.786**	.423	.057	.427					
Correct Objects	.669*	.228	-.155	.170	.910***				
Accurate Configuration	.229	.172	-.056	.124	.332	.477			
Correct Answers	.248	-.182	-.370	-.226	.536	.803*	.558*		
Example Time	.856***	.831***	.553*	.901***	.681*	.495	.217	.130	
New-Example Time	.228	-.182	-.055	-.142	.616*	.755*	.222	.793**	.301

Legend: Level of significant for one-tailed, *** $p < 0.0025$, ** $p < 0.01$ level, * $p < 0.05$ level (df=8).

6.6. Discussion

The overall aim of this experiment was to evaluate the efficacy of CD for generating program specification expressions in comparison with using natural language (NL) expressions. It was predicted that participants given the CD notation would obtain a lower proportion of correct answers in a longer training time with many returns to examples, because they had no prior experience of the CD notation. Although the CD group spent less time on the training, they obtained more correct scores, and fewer returns to the examples compared with the NL group.

The experiment took the form of a training-based experiment in which 20 participants were given instructions and training either on CD or equivalent NL specification

expressions, and then they were asked to generate specifications for the expressions, in their particular notation. Overall, we got positive results from this experiment; participants generated more accurate specifications in CD than using the other notation. It seemed that the CD notation is really an intuitive and easy to use notation, which again supports Kent's claim (Kent, 1997). It is also easy to understand its interpretation when using CD to design software systems (Howse & Schuman, 2005).

In general, Table 6.3 shows that CD is better than NL in terms of correct answers, correct objects, created objects, accurate configuration, number of returns to examples and the time spent on returned-to examples, on new examples and on examples overall. Table 6.4 shows that the CD group was not better than the NL group in terms of learning time, and steps. Also, Table 6.4 shows that CD was not better than NL in terms of initial thinking time, and steps.

During this experiment, within the NL group, one user tended to draw circles on papers. Another NL user tended to image the case by making circular diagrams using his hands to draw on the table or in the air with mental visualizations. These two attempts were done to understand the purely textual statements since the construction is a process of interpretation and translation. The NL users did not know that their colleagues were using similar diagrams in the experiments. For fair comparison, participants were randomly assigned to one group despite what representation they would have preferred. We found that the CD notation was more effective than the NL notation and also we found that there were no drop-outs in this voluntary experiment.

Many experimental design phases were made. It is a complicated design and we do not know how these decisions affected participants' performance. It may be the case that a drawing-with-paper-and-pencil environment would be faster than using a new editor which requires some time to familiarize oneself with and to learn how to use it.

Moreover, as it was a training experiment, participants were required to both learn and generate specification at the same time. Although immediate feedback was required as a training aid, there was a lack of detailed feedback. Both groups were inexperienced in using notations for specifying programs.

Table 6.3 Summary of the Results where the CD group performed better than the NL group

	Correct Objects	Accurate Configuration	Correct Answers	No. Returns	Created Objects	Example Time	New-Example Time	Returned-Example Time
CD mean (SD)	0.786 (0.060)	0.957 (0.074)	0.307 (0.461)	5.885 (10.527)	4.67 (3.35)	129.84 (96.80)	83.57 (61.98)	46.27 (66.77)
NL mean (SD)	0.759 (0.045)	0.909 (0.109)	0.157 (0.364)	9.957 (16.737)	4.67 (3.96)	133.68 (97.10)	65.51 (50.10)	68.17 (77.90)
T-Test result for individual questions or examples covering both groups	For p<.002 Q(11, 13) For p<.01: Q(3, 12) For p<.05: Q(7, 14, 21) (the CD group had more correct-objects than NL (in 13 questions))	For p<.002 Q(3, 8, 11, 14, 17) For p<.01: Q(4, 15) For p<.05: Q(1, 2, 5, 6, 9, 12, 16, 20) (In only 3 questions, the NL group had more accurate configuration than the CD)	For p<.002 Q(3, 11) For p<.01: Q(1, 14) For p<.05: Q(9,16, 19, 21) (In 16 questions, out of 21, The CD group has more correct answers than NL)	For p<.05 Q(10, 16, 18) (In 5 questions the CD group had more number of returns than the NL)	For p<.002: Q(8, 9) For p<.01: Q(14) For p<.05: Q(1, 3, 7, 10, 11,12,13, 16, 19, 20) (In 1 question the CD group had created-objects equal to the target which is better than the NL and in 19 questions fewer than NL group)	For p<.05 E(7) (The CD group achieved a time that was less than that of the NL group in all examples except examples 6 and 7)	For p<.05 E(6, 7) (The CD group achieved a time that was less than that of the NL group in two examples: E3 and E4)	For p<.01 Q(10) For p<.05 Q(16, 18) (the NL group spent more time on returned-to examples than CD)
Correlation between the groups	significant at p<0.002 (df=19)	significant at p<0.05 (df=19)	Significant (p<0.002) (df=19)	Not significant at p<0.05 (df=19)	significant at p<0.002 (df=19)	not significant at p<0.05 (df=5)	not significant at p<0.05 (df=5)	significant at p<0.05 (df=19)

Table 6.4 Summary of the Results where the NL group performed as well as or better than the CD group

	Initial Thinking	Steps
Overall CD mean (SD) Overall NL mean (SD)	24.69 (20.52) 18.80 (14.67)	1.36 (1.30) 1.32 (1.26)
T-Test result for individual questions or examples covering both groups	For $p < 0.002$: Q(1, 13) For $p < 0.05$: Q(3, 5, 7, 14) In only 4 questions the CD group needed less initial thinking time than the NL	For $p < 0.01$: Q(5, 8, 9) For $p < 0.05$: Q(2, 6, 7, 13, 14, 16) (the CD group and NL group had no significant differences in performing steps and thus the CD notation is as good as the formal NL notation)
Correlation between the groups	significant at $p < 0.002$ (for d.f.=19)	Not Significant at $p < 0.002$ for (df=19) Significant at $p < 0.0540$

The next chapter will have a summary of the previous studies, along with the overall conclusion.

Chapter 7 General Conclusions

This concluding chapter brings together the key findings and results obtained throughout the thesis. Section 7.1 gives a brief summary of the research. Section 7.2 discusses the benefits of using the proposed triangulated evidence by discussing the research findings. Finally, section 7.3 discusses the limitations of this study and gives suggestions for extending this work, as well as final thoughts on what could be improved.

7.1. Introduction

The primary aim of the research presented in this thesis was to investigate the usability of Constraint Diagrams as a program specification language by using different approaches to determine the strengths and weaknesses of that language. This research reveals that CD seems to be a promising language for use by novices who are not expert in software design, to specify programs related to their fields such as medicine, chemical engineering, etc. A series of theoretical and practical studies were conducted to investigate this usability.

Multiple methods were adopted in order to provide triangulated evidence of the potential benefits of constraint diagrams compared with other notational systems – NL in this thesis. Three main approaches were adopted for this research and they were worked together.

The first approach, which was a theoretical evaluation, was a semantic and task analysis of CD notation. This was conducted by the application of the Cognitive Dimensions of Notations framework (CogDim), which was used to examine the relative strengths and weaknesses of both CD and NL notations in terms of the perceived facilitation or impediments of these different representations. CogDim suggested five activities and for each of these we evaluated both notations according to the fourteen cognitive dimensions.

As we extensively discussed in chapter 4, CogDim is an approach that could help a designer to cognitively understand the design of a notation. According to the designer's needs, a selected dimension can be traded off with other dimensions to some degree. This framework highlights that changing a dimension will entail a change in some other

dimensions. For example, in order to reduce the degree of viscosity, we could increase the number of abstractions, which would both create hidden dependencies and reduce the visibility level, and thus increase the level of hard mental operations.

Although there is a study on the usability of constraint diagrams using CogDim (Morgan, 2011), it did not provide a full usability profile of the notation. Moreover, it adopted a different way of applying CogDim to constraint diagrams by combining novices' feedback of the interpretation of the notation to analyse dimensions.

From our systematic analysis, we found that the cost of the activities using CD was less than the cost using NL. Only in one dimension, premature commitment, did CD not satisfy two activities. However, this lack of satisfaction is needed to ensure the diagrams are formally valid. Exploratory design and incrementation activities had a high level of premature commitment, which could be considered as a problem. However, since viscosity was very low this lets premature commitment be less costly, since bad guesses can easily be corrected.

In this study, we used cognitive dimensions to examine the activities and we used these activities to find any undesired degree of the dimensions. For example, we faced a problem of high level premature commitment with the CD notation in exploratory design and incrementation activities; luckily the level of viscosity was low, which meant correction guesses would be low cost. In fact, as shown in our evaluation using CogDim, the levels of viscosity and premature commitment in the NL notation were high for exploratory design and modification activities; thus to do these two activities using this notation we could adopt a different notation such as the CD notation that had a lower degree of viscosity and then transfer the specification to the target notation, which breaks this process into two activities: exploratory design activity using CD notation followed by a transcription activity to transfer from CD to NL.

In general, Cognitive Dimensions of Notations helped us in understanding the nature of the structure of the CD notation. Despite the fact that CD is a complex notation, this framework is suitable for exploring the cognitive aspects of this complex notation. This study provides a full profile for each notational activity supported by a number of cognitive dimensions, which could help in understanding how such a notation can be used in specifying software.

The second approach, which is the first empirical experiment, was conducted to examine the interpretation of CD in order to evaluate users' comprehension of this notational system. This study was based on comparing the efficacy of CD and NL for understanding program specification statements. It was also a practical evaluation of the searching activity used in chapter 4.

This experiment took the form of a web-based competition in which 33 participants were given instructions and training on either CD or equivalent NL specification expressions, and after each training example, they responded to three multiple-choice questions requiring the interpretation of expressions in their particular notation. Participants had no prior experience in applying new notations neither in program specification nor in CD notation.

In the outcomes of this experiment we found that the CD group spent more time on the training and had lower confidence. However, they obtained comparable interpretation scores to the NL group and took less time to answer the questions, although they had no prior experience of CD notation.

Overall, using CD even by novice users for interpreting a specification for software can save time spent on the searching for information activity that is needed for interpretation. CD novice users showed accurate understanding of the notation. Thus, after CD users had been trained at using CD, they found this notation effective in terms of interpretation time.

The third approach, which was the second empirical experiment, was conducted to examine the construction of program specifications. This experiment was based on comparing the efficacy of CD with NL for generation of program specification statements. This experiment evaluated the use of CD to build program specifications for a health informatics case study called a Patient Record System (Fetats, et al., 2005). In this experiment, we also evaluated the five activities mentioned in chapter 4: exploratory design, modification, incrementation, searching, and transcription activities.

In this experiment, which focused on the construction of CD, 20 participants were given instructions and training on either CD or equivalent NL specification expressions. After each training example, they responded to three questions requiring the construction of expressions in their particular notation. We built an editor that allowed construction in

the two notations, which automatically logged their interactions. Participants had no prior experience in applying new notations in program specification, or in CD notation.

From this experiment, we found that although the CD group had more accurate answers, they spent more time answering the questions. The NL group gave answers that were partially correct but with some missing information. Moreover, the CD group had spent more time in training, but their returns to the training examples were fewer than in the case of the NL group.

According to the construction of the program specification, CD novice users showed accurate understanding of using the notation to construct the required specification. However, due to the lack of familiarity with both CD notation and program specification, CD users needed more time for training and constructing the specification.

In general, according to CogDim analysis and both interpretation and construction experiments, it has been shown that constraint diagrams are clearly a promising language for program specification in contrast with a natural language approach. The complexity of the problems represented requires a notation that simplifies them and explores unobvious constraints that should be known during the program specification before the implementation of the program. It is widely accepted that describing constraints in natural language will always result in ambiguities, and when describing constraints in traditional formal language, this will always need a strong mathematical background.

We will now outline the implications they bring for program specifications, along with the limitations and the final views on future recommendations.

7.2. Implications of the Key Findings

In the introduction chapter, five questions were posed:

1. Is CD notation effective for program specification?
2. Is CD a good notation for the construction of program specification expressions?
3. Is CD a good notation for the interpretation of program specification expressions?

4. What are the relative strengths and weaknesses of CD notation and conventional NL notation?
5. Is CD notation effective for supporting novice users?

The objective of this research was to provide a theoretical evaluation and two practical experiments, which is triangulated evidence of the potential benefits of CD notation compared with NL notation to answer to these questions. We will address the questions in reverse order and will discuss them in the following subsections.

7.2.1. Is CD notation effective for supporting novice users?

Novice users who are new to program specification and new to using the CD notation did not find any difficulties in learning this notation. In both the interpretation and construction experiments, the item analysis showed that there was little difference between the two groups, even on the most difficult questions.

Furthermore, there was no significant difference in the time taken for learning CD and NL notations, despite the fact that the NL users were already familiar with the NL notation. Use of CD notation rather than NL notation even by novice users for interpreting a specification for software saved time spent on searching for information that was needed for interpretation. In general, CD novice users showed accurate understanding of using the notation to both interpret and construct the required program specification. Thus, the CD notation is effective for supporting novices in at least two aspects of specifying programs.

7.2.2. What are the relative strengths and weaknesses of CD notation and conventional NL notation?

On one hand, in terms of the cognitive facilitation, CD notation is cognitively less costly compared to natural language. Unlike NL notation, CD notation has free rides and thus hard mental operations occur less than with NL notation. Moreover, the CD notation has a high degree of role-expressiveness which could help novice users to easily infer the meaning of the purpose of the components used. Furthermore, since related information is visually grouped, errors can be easily spotted. In general, according to the cognitive dimensions, CD notation has the required degree of the dimensions for the activities. For example there is no need to reduce the degree of viscosity, and thus we will not increase the amount of abstraction, which will not create hidden dependencies nor reduce the visibility level, and thus, will not increase the level

of hard mental operations. In general, the cognitive cost of the program specification activities using CD notation is less than the cost using NL notation. In Chapter 4 we discussed the cognitive strengths that CD notation has and how the cost of it is less than that of natural language.

On the other hand, in terms of the cognitive impediments, the CD notation has a problem of a high level of premature commitment which could force users to think ahead and make certain decisions when used in exploratory design and incrementation activities. Although this enforcement could be considered as a cognitive weakness, it could be viewed as a good way to support the validation of these notation diagrams, and luckily the level of viscosity of CD notation was low, which means correction guesses were at no cost.

Furthermore, in terms of the comprehension and construction impediments, due to the unfamiliarity with both CD notation and program specification, CD users needed more time for training and constructing the specification. However, when users were trained on CD notation, it was a faster tool for interpretation than NL notation.

Overall, the CD notation was cognitively effective for program specification.

7.2.3. Is CD a good notation for the interpretation of program specification expressions?

As shown in the previous two answers, CD notation supports novice users and also is cognitively effective for program specification. This is an indication that CD notation is effective enough to play a role. So we examined this indication with the interpretation role in this case.

From the interpretation experiment, we found that the CD notation was better than the NL notation for interpretation tasks done by novice users. Overall, it is effective for understanding the program specification accurately and it can actually be used to save time spent on interpreting the program specification. Despite the fact that NL notation is easy to use for specification, especially for narrative descriptions, it has the problem of being ambiguous and allowing different inferences. However, CD notation, due to its ability to visualize the properties of relationships and the relative positions of the elements, such as being in a subset of other sets, and also due to having a free-ride property, is better to use because it is not only a visual but also a formal language.

Overall, there were no particular types of concept that showed difficulties, even after Bonferroni corrections.

7.2.4. Is CD a good notation for the construction of program specification expressions?

As shown in the previous answer, there is an indication that CD notation is effective enough to play an interpretation role. In this case, we will examine this indication with another role in program specification, which is the construction role.

From the construction experiment we found that the CD notation was better than the NL notation for construction tasks done by novice users. Overall, due to the fact that CD notation can compact information into a small space, it is more effective than NL notation in constructing a program specification accurately. In general, there were no particular types of concept that showed difficulties, even after Bonferroni corrections.

7.2.5. Is CD notation effective for program specification?

As shown in the previous two answers, CD notation supports novice users and also is cognitively effective for program specification. It is also a good notation for both interpreting and constructing program specification.

As a conclusion from this study, we found that CD is as effective as NL, which is a familiar language that novice users would always use. Overall, Kent claimed that CD could be used to bridge the gap between formal and informal specification languages by providing an intuitive and expressive unambiguous diagrammatic formal language, and that it is simpler and more effective than other approaches used to formally specify programs (Kent, 1997); it also has been claimed it could be used for software specification (Howse & Schuman, 2005; Fetais, et al., 2005); and it has been claimed that its interpretations are not always well matched to the meaning (Stapleton & Delaney, 2008). We supported these claims by using triangulated evidence and we went further than that by showing that CD notation is not only better to use for interpretation tasks but also better for construction tasks than NL notation, which is a good first step to evaluating how CD notation could be used in program specification. Although there was an overhead cost of learning the CD notation for interpretation, this cost was not too great, and it was less than the cost of NL notation for construction tasks. Thus, hopefully constraint diagrams will be used by non-computer scientists to bridge the gap

in communication between software designers and stakeholders when specifying programs in order to be accurate and concise.

Overall, it seems that CD is intuitive and expressive, with an unambiguous semantic notation, which supports Kent's claim (Kent, 1997). It is also easy to understand its interpretation when using CD to design software systems. However, the proportion of correct answers and the confidence level on answering indicate that the interpretations were not always well-matched to the meaning (Stapleton & Delaney, 2008).

7.3. Thesis Limitations and Future Work

We consider this thesis as the first step in evaluating the usability of CD notation by comparing it with natural language using theoretical and empirical methods. The theoretical method was an analysis which was obtained from the Cognitive Dimensions of Notations framework. Although Green did not explicitly claim that CogDim was objective in its conclusions, there is still an issue of how objective it is. Overall, we found that CogDim is subjective, because the analysis in Chapter 4 was based on the author's beliefs and experience in both CD and NL notations. Thus, we believe that this framework should include, besides the activities and the dimensions, the experience level of the users, which would enrich the analysis with the cognitive strengths and weaknesses of such notations from the point of view of both the expert and the novice user. For example, cognitively a novice user could consider the CD notation as having a high degree of hidden dependencies because of the reading issues, but an expert may consider that the degree is low because of the generalized constraint diagrams notation. Both of them are right for the reasons they have given, which is an indication of their expertise level in that notation. Another example: novice users who want to denote the existence of a spider (an element) without knowing the exact place may consider putting the spider on the edge of the contour as an option. However, an expert would consider this a visual error because he knows that having unknown places means that such a spider would have feet and legs. The CogDim is also based on the selection of examples used, whether they are easy or hard, and the case studies on whether they are simple or complex. Moreover, this framework required every dimension to be described (Blackwell, et al., 2001) with illustrative examples, case studies, and associated advice for designers. Thus, the chosen selection of simple or complex examples and case studies will definitely affect the analysis, and thus this framework should include, besides the activity, the dimensions and the experience level of the users, and the level

of both examples and case studies. Furthermore, although there are definitions of the cognitive dimensions, there is no standard definition of their scale – of the degree of each dimension. For example, the degree of hidden dependencies for the novice user is high, but for the expert it is low. The question here is how high the degree is, and how low the degree is; and also when we could consider the degree as medium. In general, without having a bar, it was difficult to judge the degree of a cognitive dimension without being subjective, which we believe is a major flaw of the CogDim framework. In general, we disagree with Green that the CogDim framework is a usability evaluation technique (Green, 1996) because it is only a “check list” and a “discussion tool” (Green, 1996) with other evaluators. Furthermore, since the target for any usability study is the users who will use such notations, we have to disagree with Green and his colleagues about limiting the scope of users to focus only on designers (Blackwell, 2001). Thus, as a recommendation, we think several people, experts and non-experts, using both CD and NL languages, are required to apply different dimensions to a variety of tasks. However, this will be costly, which is against the notion of proposing CogDim as being “extremely quick and cheap” (Green, 1996). In fact, as it is, we did not find it “quick and cheap” because it took time to understand the definition of each of the cognitive dimensions, to understand the dimensions of the dimensions, to understand the relationship between different dimensions, and to understand the differences between the dimensions when applying it to different notational activities. Overall, we found that this framework, despite its observed limitations, was an effective tool to provide a fair sense of how the notation would react to a notational activity, to provide more understanding of the useful aspects of the notation, and to provide a standard vocabulary that could be used when questioning others about certain notational features.

For the empirical experiments, the number of participants was limited. In our interpretation experiment, which was an online experiment, 28 of the 53 participants dropped out, while the drop-out rate was zero in the construction experiment. Perhaps the reason was that the latter experiment was conducted in the lab, which was a controlled setting and thus affected the rate. Overall, although we found from the empirical studies that the CD notation requires more time to learn how to interpret, we found it required less time to learn how to construct compared with the NL notation. In general for both experiments the NL notation did not provide more accurate answers

than the CD notation. For these reasons the CD notation was better than the NL notation.

Although we empirically evaluated the CD and NL notations with novice users, who had no, or relatively little, background of either program specification or using CD notation, the research should be extended to compare an experienced CD group with an experienced NL group. This could provide us with the usability of such notations from different points of view and we could use the results from the proposed study to compare them with our present results and get a wide-ranging indication of usability. Indeed in our study the group was experienced in NL, but not in using NL for specifying programs.

The experiment's design might also be improved. As they were training experiments, participants had to both learn and perform a task at the same time. Perhaps we could separate the learning and the performing tasks and see how spending more hours learning the CD notation and using it would affect the results. The interpretation experiment was intended to last for an hour and the construction experiment was intended to last for two hours. However, if we let participants learn and use the CD notation for 10 hours with different trails, it may change the results. In fact, we found in the interpretation experiment that participants tended not to return to examples. We would like to believe that they might understand the notation well enough just from one reading. However, due to the fact that it was a competition and the time taken was recorded, we have to assume that they might have decided to save time by not returning to examples.

The learning in our experiments depended on both the possibility to return to the related example at any time while answering and the immediate feedback on the answers. We found that the NL group returned to the examples more than the CD group, which meant they needed help in finding out how to solve the questions. This might indicate that the specification of a model using the CD notation was more memorable than by using the NL notation. In general, we would like to provide participants with detailed feedback and check whether this would affect the drop-out rate or not.

The material we used covered selected concepts of CD notation and thus the range or scope of the examples used and their complexity are relatively limited. We aimed at evaluating the basic concepts of CD notation because we focused on understanding how

users, especially novices, would understand and use simple program specifications. Kosslyn (Kosslyn, 1989; Kosslyn & et al., 1990) found that increased visual data complexity would reduce comprehension. Thus, we tended to make it as simple as possible by avoiding the use of complex examples where a reading tree must be explicit. Also, we tended to include only the basic concepts of CD notation and not the full version of the relations between spiders such as the strand.

We used a fragment of the Patient Record System and thus it was not used as a whole system. As mentioned previously, we were evaluating the usability of this notation for novice users and thus we tended to cover the basic concepts of CD notation and a simple case study system. We would recommend repeating this study with different case studies which could be real-world case studies taken from companies that are interested in software designs. Our research depended on theoretical case studies which might or might not apply in the real world, and thus it could be considered as one example of a follow-up study.

Choosing to use multiple choice rather than free-form open-ended responses could affect the results. The multiple choice forced participants to pick one answer, which could result in providing a vague interpretation. Perhaps translation tasks would produce clearer interpretations.

The construction of the CD notation experiment did not involve free-hand drawing. It was an easy drag-and-drop construction of diagrams task to eliminate the chances of human error during drawing. On one hand, we limited the users' creativity by forcing them to use a specific number of shapes by constraining their available tools. On the other hand, despite the fact that free-hand drawing would not constrain their options, it might result in drawing ambiguous diagrams. Thus we would still recommend the drag-and-drop feature for construction tasks.

The interpretation and construction tasks were relatively simple. For example, participants were provided with one invariant at a time. In our evaluation, we aimed at simple tasks because our research was a first step in studying the usability of CD notation. We could examine participants' interaction by providing them with many related invariants at once, which might produce different results from ours. In general, this research had only one experiment in interpretation and only one experiment in construction. Perhaps conducting more experiments would be beneficial.

This construction experiment was about creating diagrams. However, there were no practical studies on tasks such as modifying existing modelling using the CD notation. We would like to conduct some studies to examine how the use of CD notation as a construction tool might affect the programming implementation phase; why CD notation would allow people to accomplish tasks faster than purely textual notations; and how CD notation would affect the software lifecycle.

Furthermore, we used general measurements to evaluate the performance, such as the time spent on learning and using the notation, and the percentage of accurate answers, as well as subjective measurement of the confidence rating. However, we do not know whether the user was spending time on learning how to use the notation or learning how to use the editor. Thus, we may need to understand the user interactions with the design of the notation and distinguish between the time spent using the notation and time spent using the editor by measuring whether the user was reading, searching or scanning information, or simply looking at the screen. This could be conducted by eye-tracker experiments. These experiments could add detailed data to the usability studies, which would provide a full understanding of this notational system and extend this study to understand the nature of CD notation.

Moreover, our editor, which took a long time to construct, could be improved by having an intelligent electronic answer-marking schema to record the validity of the diagrams by differentiating between legal diagrams, correct-answer diagrams, and partially correct or legal diagrams, which would be a benefit. Indeed, the way in which our developed editor worked for the two notations may impact on the performance.

Furthermore, we compared this notation with natural language for the reasons discussed in Chapter 1. However, it would be beneficial to compare it with other languages for different reasons. To elaborate, we could compare it with another diagrammatic language such as UML (Chapter 2, section 2.3) or to a formal language such as Z or OCL (Chapter 2, section 2.2). Indeed, the contrast with the formal languages would be interesting to explore because they all share the same properties of being formal and could be applied for program specification.

Bibliography

- Barwise, J. and Etchemendy, J. (1995) Heterogenous logic. In Glasgow, J., Narayanan, N. H., and Chandrasekaran, B. (eds.) *Diagrammatic Reasoning: Cognitive and Computational Perspectives*. Menlo Park, CA: AAAI Press: 211-234.
- Bevan, N. (1997) Quality and Usability: a New Framework. In Van Veenendaal, E., and McMullan, J. (eds.) *Achieving Software Product Quality*, Tutein Nolthenius, Netherlands.
- Blackwell, A. (1997) Diagrams about Thoughts about Thoughts about Diagrams. In Anderson, M. (ed.) *Reasoning with Diagrammatic Representations II: Papers from the AAAI 1997 Fall Symposium*. Technical Report FS-97-02. Menlo Park, California: AAAI Press: pp. 77-84.
- Blackwell, A. and Green, T.R.G. (2003) Notational Systems – the Cognitive Dimensions of Notations Framework. In Carroll, J. M. (ed.) *HCI Models, Theories, and Frameworks: Toward a Multidisciplinary Science*. San Francisco, Morgan Kaufmann: 103-134.
- Blackwell, A.F., Britton, C., Cox, A. Green, T.R.G., Gurr, C.A., Kadoda, G.F., Kutar, M., Loomes, M., Nehaniv, C.L., Petre, M., Roast, C., Roes, C., Wong, A. and Young, R.M. (2001) Cognitive Dimensions of Notations: Design Tools for Cognitive Technology. In Beynon, M., Nehaniv, C.L. and Dautenhahn, K. (eds.) *Cognitive Technology 2001 (LNAI 2117)*. Springer-Verlag, pp. 325-341.
- Blackwell, A.F., Green, T.R.G. and Nunn, D.J.E. (2000) Cognitive Dimensions and Musical Notation Systems. Paper presented at ICMC 2000, Berlin: Workshop on Notation and Music Information Retrieval in the Computer Age.
- Booch, G., Rumbaugh, J. and Jacobson, I. (1999) *The Unified Modeling Language User Guide*. Second Edition, Addison Wesley.
- Bottoni, P., Koch, M., Parisi-Presicce, F. and Taentzer, G. (2001) A Visualization of OCL Using Collaborations. In Gogolla, M., Kobryn, C. (eds.) *UML 2001*. LNCS, vol. 2185, pp. 257-271. Springer, Heidelberg.
- Burns, C. M. and Hajdukiewicz, J. R. (2004) *Ecological Interface Design (EID)*. Boca Raton: CRC Press.
- Burton, J. (2011) *Generalized Constraint Diagrams: the Classical Decision Problem in a Diagrammatic Reasoning System*. PhD thesis, University of Brighton.
- Card, S. T., Moran, T. P. and Newell, A. (1983) *The Psychology of Human-Computer Interaction*. Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.
- Card, S., Mackinlay, J. and Shneiderman, B. (1999) *Readings in Information Visualization. Using Vision to Think*. San Francisco: Morgan Kaufmann Publishers.
- Carroll, J. B. (ed.) (1956) *Language, Thought, and Reality: Selected Writings of Benjamin Lee Whorf*, The MIT Press.

- Chen, P. (1976) The Entity-Relationship Model - Toward a Unified View of Data. *ACM Transactions on Database Systems* 1 (1): 9-36.
- Chen, P. (2002) Entity-Relationship Modeling: Historical Events, Future Trends, and Lessons Learned. In *Software pioneers*, pp. 296-310, Springer-Verlag.
- Cheng, P. C. H. (1996) Scientific Discovery with Law-Encoding Diagrams. *Creativity Research Journal* 9(2-3): 145-162.
- Cheng, P. C. H. (2002) Electrifying Diagrams for Learning: Principles for Complex Representational Systems. *Cognitive Science* 26(6): 685-736.
- Cheng, P. C. H. (2003) Diagrammatic Re-Codification of Probability Theory: A Representational Epistemological Study. The twenty fifth annual conference of the cognitive science society, Boston, MA, Cognitive Science Society.
- Cheng, P. C. H. and Shipstone, D. M. (2003) Supporting Learning and Promoting Conceptual Change with Box and AVOW Diagrams. Part 2: Their Impact on Student Learning at A-level. *International Journal of Science Education* 25(3): 291-305.
- Cheng, P. C.-H. and Barone, R. (2007) Representing Complex Problems: A Representational Epistemic Approach. In Jonassen, D. H. (ed.), *Learning to Solve Complex Scientific Problems*. Mahmah, N.J.: Lawrence Erlbaum Associates.: 97-130.
- Cheng, P. C.H. and Simon, H. A. (1992) The Right Representation for Discovery: Finding the Conservation of Momentum. In Sleeman, D. and Edwards, P. (eds.), *Machine learning: Proceedings of the ninth international conference (ml92)* (pp. 62-71). San Mateo, CA: Morgan Kaufmann.
- Chow, S. and Ruskey, F. (2003) Drawing Area-Proportional Venn and Euler Diagrams. In *Proceedings of Graph Drawing 2003*, Perugia, Italy, Springer-Verlag 2912 of LNCS: 466-477.
- Churchill, G. A. J. (1979) A Paradigm for Developing Better Measures of Marketing Constructs. *Journal of Marketing Research* 16(1): 64-73.
- Clark. R. (2005) Failure Mode Modular De-Composition Using Spider Diagrams. *Proceedings of Euler Diagrams 2004 Elsevier, ENTCS vol. 134*, pages 19-31.
- Clarke, S. (2001) Evaluating a New Programming Language. 13th Workshop of the Psychology of Programming Interest Group, Bournemouth, UK: 275-289.
- Clarke, S. (2004) Measuring API Usability. *Dr. Dobb's Journal Special Windows/.NET Supplement*, 5, S6-S9.
- Clarke, S. and Becker, C. (2003) Using the Cognitive Dimensions Framework to Measure the Usability of a Class Library. *Proceedings of the First Joint Conference of EASE PPIG (PPIG 15)*
- Cook, S. and Daniels, J. (1994) *Designing Object Systems: Object-Oriented Modelling with Syntropy*. Prentice Hall Object-Oriented Series.

- Cox, R. (1996) Analytical Reasoning with Multiple External Representations. Artificial Intelligence, University of Edinburgh. PhD thesis.
- Cox, R. (1997) Representation Interpretation versus Representation Construction: An ILE-based Study using SwitchERII. Artificial Intelligence in Education: Knowledge and media in learning systems (Proceedings of the 8th World Conference of the Artificial Intelligence in Education Society). Boulay, B. and Mizoguchi, R. Amsterdam, IOS: 434-441.
- Cox, R. and Brna, P. (1995) Supporting the Use of External Representations in Problem Solving: the Need for Flexible Learning Environments. Journal of Artificial Intelligence in Education 6(2): 239-302.
- DeChiara, R. and Fish, A. (2007) EulerView: A Non-Hierarchical Visualisation Component. In Proceedings of IEEE Symposium on Visual Languages and Human Centric Computing IEEE 134: 145-152.
- DeChiara, R., Erra, U. and Scarano, V. (2003) VennFS: A Venn Diagram File Manager. In Proceedings of Information Visualisation, IEEE Computer Society: 120-126.
- Dreyfus, T. (1994) Imagery and Reasoning in Mathematics and Mathematics Education. Selected Lectures from the 7th International Congress on Mathematical Education. Robitaille, D.F., Wheeler, D. H. and Kieran, C. Quebec, Canada, Le Presses de l'Université Laval: 107-122.
- D'Souza, D. and Wills, A. (1995) Catalysis: Practical Rigour and Refinement. Technical report, ICON Computing.
- D'Souza, D. and Wills, A. (1998) Objects, Components, and Frameworks With UML: The Catalysis Approach. Addison-Wesley.
- Eilenberg, S. and MacLane, S. (1945) General Theory of Natural Equivalences. Transactions of the American Mathematical Society 58(2): 231-294.
- Fetais, N., Howse, J., Schuman, S. (2005) Patient Record System. MSc thesis, University of Brighton.
- Fish, A. and Howse, J. (2003) Computing Reading Trees for Constraint Diagrams. In AGTIVE'03, Applications of Graph Transformations with Industrial Relevance, Charlottesville, Virginia, Springer-Verlag, pages 260-274.
- Fish, A. and Masthoff, J. (2005) An Experimental Study into the Default Reading of Constraint Diagrams. VLHCC05, Proceedings of Visual Languages and Human Centric Computing, Dallas: 287 – 289.
- Fish, A., Flower, J. (2005) Investigating Reasoning with Constraint Diagrams. Electronic Notes in Theoretical Computer Science 127: 53-69.
- Fish, A., Flower, J. and Howse, J. (2005b) The Semantics of Augmented Constraint Diagrams. Journal of Visual Languages and Computing 16(6): 541-573.

- Fish, A., Flower, J., Howse, J. (2003) A Reading Algorithm for Constraint Diagrams. In: IEEE Symposium on Human Centric Computing Languages and Environments, Auckland, New Zealand, pp. 161-168. IEEE, Los Alamitos.
- Fish, A., Howse, J. (2004) Towards a Default Reading for Constraint Diagrams. In: Blackwell, A.F., Marriott, K., Shimojima, A. (eds.) Diagrams 2004. LNCS (LNAI), vol. 2980, pp. 51-65. Springer, Heidelberg.
- Fish, A., Howse, J., Taentzer, G. and Winkelmann, J. (2005a) Two Visualizations of OCL: A Comparison. Technical Report VMG.05.1, University of Brighton.
- Fish, A., Khazaei, B. and Roast, C. (2011) User-comprehension of Euler diagrams. *Journal of Visual Languages and Computing* 22(5): 340-354.
- Flower, J. and Stapleton, G. (2004) Automated Theorem Proving with Spider Diagrams. *Electronic Notes in Theoretical Computer Science* 91(0): 246-263.
- Frandsen, A. N. and Holder, J. R. (1969) Spatial Visualization in Solving Complex Verbal Problems. *Journal of Psychology* 73(2): 229-233.
- Gil, J. and Sorkin, Y. (2013) The Constraint Diagrams Editor. Available at www.cs.technion.ac.il/Labs/ssdl/research/cdeditor/. [August 2013]
- Gil, J., Howse, J. and Kent, S. (1999) Formalising Spider Diagrams. In *Proceedings of IEEE Symposium on Visual Languages (VL 1999)*, Tokyo, pp. 130-137. IEEE Computer Society Press, Los Alamitos.
- Gil, J., Howse, J. and Kent, S. (2001) Towards a Formalization of Constraint Diagrams. *Human-Centric Computing Languages and Environments*, 2001. *Proceedings IEEE Symposia on on Human Centric Computing Languages and Environments*, Stresa.
- Gil, J., HOWSE, J. and Tulchinsky, E. (2002) Positive Semantics of Projections in Venn-Euler Diagrams. *Journal of Visual Languages and Computing*, 13 (2). pp. 197-227.
- Gil, Y. and Kent, S. (1998) Three Dimensional Software Modelling. *Proceedings of International Conference in Software Engineering 1998*, IEEE Computer Society Press.
- Glasgow, J., Narayanan, N. H. and Chandrasekaran, B., (eds.) (1995) *Diagrammatic Reasoning: Cognitive and Computational Perspectives*, Cambridge, Mass: The MIT Press and AAAI Press.
- Glinert, E. (1989) Towards a Software Metrics for Visual Programming. *International Journal of Man-Machine Studies* 30: 425-445.
- Goldson, D., Reeves, S. and Bornat, R. (1993) A Review of Several Programs for the Teaching of Logic. *The Computer Journal* 36(4): 373-386.
- Green, T. R. G. (1989) Cognitive dimensions of Notations. In *People and Computers*, cambridge, Cambridge University Press.
- Green, T. R. G. (2000) Instructions and Descriptions: Some Cognitive Aspects of Programming and Similar Activities. Invited paper, in Di Gesù, V., Levialdi, S. and Tarantino, L.,

- (eds.) Proceedings of Working Conference on Advanced Visual Interfaces (AVI 2000). New York: ACM Press: 21-28.
- Green, T. R. G. and Petre, M. (1996) Usability Analysis of Visual Programming Environments: A 'Cognitive Dimensions' Framework. *Journal of Visual Languages and Computing* 7(2): 131-174.
- Green, T.R.G. (2006) Aims, Achievements, Agenda – Where CDs Stand Now. *Journal of Visual Languages and Computing* 17(4): 288-291.
- Green, T.R.G. and Blackwell, A. F. (1998) Cognitive Dimensions of Information Artefacts: A Tutorial. Version 1.2.
- Green, T.R.G., Blandford, A.E., Church, L., Roast, C.R. and Clarke, S. (2006) Cognitive Dimensions: Achievements, New Directions, and Open Questions. *Journal of Visual Languages and Computing*. 17(4), 328-365.
- Gruber, T. R. (1993) A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition* 5(2): 199-220.
- Gurr, C. and Tournas, K. (2000) Towards the Principled Design of Software Engineering Diagrams. In *Proceedings ICSE 2000: 22nd International Conference on Software Engineering*, pages 509–518. ACM Press.
- Halpin, T., Evans, K., Hallock, P. and MacLean B. (2003) Database Modeling With Microsoft Visio for Enterprise Architects, Morgan Kaufmann Publishers: San Francisco.
- Hammer, E. and Danner, N. (1996) Towards a Model Theory of Diagrams. *Journal of Philosophy Logic*, 25 (5):463-482.
- Hammer, E. M. (1995) *Logic and Visual Information*. CSLI Publications.
- Harel, D. (1987) Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming* 8: 231-274.
- Harel, D. (1988) On Visual Formalisms. *Communications of ACM*. 31(5): 514-530.
- Harel, D. (2007) Statecharts in the Making: A Personal Account. *Proc. 3rd ACM SIGPLAN History of Programming Languages Conference (HOPL III)*.
- Harel, D. and Gery, E. (1997) Executable Object Modeling with Statecharts. *Computer* 30:7, IEEE Press, 31-42.
- Harley, T. (2008) *The Psychology of Language: From Data to Theory*. Hove, UK, Psychology Press.
- Hayes, P., Eskridge, T. and Mehrotra, M. (2005) Collaborative Knowledge Capture in Ontologies. In *Proceedings of the 3rd International Conference on Knowledge Capture*: 99-106.
- Hendry, D. (1995) Display-Based Problems in Spreadsheets: A Critical Incident and a Design Remedy. 1995 IEEE Symposium on Visual Languages, Darmstadt, Germany: 284-290.

- Howse, J. and Schuman, S. (2005) Precise Visual Modelling. *Software and Systems Modelling*, 4: 310-325.
- Howse, J., G. Stapleton, Taylor, K. and Chapman, P. (2011) Visualizing Ontologies: A Case Study. *International Semantic Web Conference 1*: 257-272.
- Howse, J., Molina, F. and Taylor, J. (2000a) SD2: A Sound and Complete Diagrammatic Reasoning System. In *Proceedings VL 2000: IEEE Symposium on Visual Languages*, Seattle, USA, pages 127-136. IEEE Computer Society Press.
- Howse, J., Molina, F. and Tylor, J. (2000b) On the Completeness and Expressiveness of Spider Diagram Systems. *Theory and Application of Diagrams, Proceedings*. 1889: 26-41.
- Howse, J., Molina, F. and Tylor, J. (2000c) A Sound AND Complete Diagrammatic Reasoning System. *Proceedings. ASC 2000: 3rd IASTED International Conference on Artificial Intelligence and Soft Computing, Banff, IASTED/ACTA Press 2000*, 402-408.
- Howse, J., Molina, F., Taylor, J. and Kent, S. (1999) Reasoning with Spider Diagrams. In: *Proceedings of IEEE Symposium on Visual Languages (VL 1999)*, Tokyo, pp. 138-147. IEEE Computer Society Press, Los Alamitos.
- Howse, J., Molina, F., Taylor, J., Kent, S. and Gil, J. (2001) Spider Diagrams: A Diagrammatic Reasoning System. *Journal of Visual Languages and Computing* 12(3), 299-324.
- Howse, J., Stapleton, G. and Oliver, I. (2010) Visual Reasoning about Ontologies. In: *Proceedings of the 9th International Semantic Web Conference (ISWC2010)*, CEUR, Shanghai, China. 658: 5-8.
- Howse, J., Stapleton, G. and Taylor, J. (2005) Spider Diagrams. *LMS Journal of Computation and Mathematics*, 8: 145-194.
- Howse, J., Stapleton, G., Flower, J. and Taylor, J. (2002) Corresponding Regions in Euler Diagrams. *Diagrammatic Representation and Inference*. 2317: 76-90.
- Kent, S. (1997) Constraint Diagrams: Visualizing Invariants in Object Oriented Modelling In *Proceedings of OOPSLA97*, pages 327-341. ACM Press.
- Kestler, H., Muller, A., Gress, T. and Buchholz, M. (2005) Generalized Venn Diagrams: A New Method for Visualizing Complex Genetic Set Relations, *Journal of Bioinformatics* 21 (8): 1592-1595.
- Kiesner, C., Taentzer, G., and Winkelmann, J. (2002) VisualOCL: A Visual Notation of the Object Constraint Language. Technical Report 23, Computer Science Department of the Technical University of Berlin, Germany.
- Koedinger, K. R. (1992) Emergent Properties and Structural Constraints: Advantages of Diagrammatic Representations for Reasoning and Learning. In Narayanan, N. H. (ed.), *AAAI Technical Report on Reasoning with Diagrammatic Representations (SS-92-02)*. Menlo Park, CA: AAAI.

- Kosslyn, S. (1989) Understanding Charts and Graphs. *Applied Cognitive Psychology*, 3, 185-226.
- Kosslyn, S. M., Chabris, C. F. and Hamilton, S. E. (1990) Designing for the Mind: Five Psychological Principles of Articulate Graphics. *Multimedia Review*: 23-29.
- Langley, P. and Rogers, S. (2005) An Extended Theory of Human Problem Solving. In *Proceedings of the Twenty-seventh Annual Meeting of the Cognitive Science Society*, Stresa, Italy.
- Langley, P., Simon, H. A., Bradshaw, G. and Zytkow, J. (1987) *Scientific Discovery: Computation Explorations of the Creative Processes*. Cambridge, MA: MIT Press. Lull, R. *Ars Magma*. Lyons, 1517.
- Larkin, J. H. and Simon, H. A. (1987) Why a Diagram is (Sometimes) Worth 10000 Words. *Cognitive Science* 11(1): 65-99.
- Larkin, J., McDermott, J. Simon, D. P. and Simon, H. A. (1980) Expert and Novice Performance in Solving Physics Problems. *Science* 208: 1335-1342.
- Larman, C. (2001) *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process* Prentice Hall.
- Lewis, C., Polson, P., Wharton, C. and Rieman, J. (1990) Testing a Walkthrough Methodology for Theory-Based Design of Walk-Up-and-Use Interfaces. *CHI '90 Proceedings*: 235-242.
- Lövdahl, J. (2002) *Towards a Visual Editing Environment for the Languages of the Semantic Web*. Ph.D. Thesis, Linköping University.
- Margaris, A. (1990) *First Order Mathematical Logic*, Dover Publications.
- Modugno, F., Green, T. R. and Myers, B. (1994) *Visual Programming in a Visual Domain: A Case Study of Cognitive Dimensions*. People and Computers IX G. Cockton, S. Draper and G. Weir. Cambridge, UK, Cambridge University Press.
- Morgan, N. (2011) *The Usability of Constraint Diagrams*. MPhil Thesis, the University of Brighton.
- Nielsen, J. (1992) Finding Usability Problems through Heuristic Evaluation. In: *ACM SIGCHI 1992*: 373-380.
- Nielsen, J. and Molich, R. (1990) Heuristic Evaluation of User Interfaces. In: *ACM SIGCHI 1990*, Seattle, Washington: 249-256.
- Nilson, G.M., Hagen, H. and Muller, H., (eds.) (1997) *Scientific Visualization: Overviews, Methodologies, and Techniques*. Los Alamitos, CA:IEEE Computer Society Press.
- O'Brien, L. and Booch (2009) *G. Grady Booch on Design Patterns, OOP, and Coffee*. Pearson.
- Oestereich, B. (2002) *Developing Software with UML: Object-Oriented Analysis and Design in Practice*. Addison-Wesley Professional.

- Oliver, I., Howse, J., Stapleton, G., Nuutila, E. and Torma, S. (2009) Visualizing and Specifying Ontologies Using Diagrammatic Logics. In: Proceedings of the 5th Australasian Ontologies Workshop, CRPIT: 37-47.
- Palmer, S. E. (1978) Fundamental Aspects of Cognitive Representation. *Cognition and Categorization*. Rosch, E. and Lloyd, B. B. Hillsdale, N.J., Lawrence Erlbaum: 259-303.
- Pereira M.J., Mernik M., Cruz D., Henriques P. (2008) Program Comprehension for Domain-Specific Languages, CoRTA'08 - Compilers, Related Technologies and Applications, Julho.
- Roast, C. R. and Siddiqi, J. I. (1996) The Formal Examination of Cognitive Dimensions. *Industry Day and Adjunct Proceedings HCI '96*. Blandford, A. and Thimbleby, H. London, UK.
- Sears, A. (1993) Layout Appropriateness: A Metric for Evaluating User Interface Widget Layout. *IEEE Transactions on Software Engineering* 19: 707-719.
- Shimojima, A. (1996) Operational Constraints in Diagrammatic Reasoning. In *Logical Reasoning with Diagrams*, G. Allwein and J. Barwise, Editors. Oxford University Press: New York. p.27-48.
- Shimojima, A. (2004) Inferential and Expressive Capacities of Graphical Representations: Survey and Some Generalizations. In *Diagrams*, vol 2980 of LNAI, Springer, pp 18-21.
- Shimojima, A. and Katagiri, Y. (2008) An Eye-Tracking Study of Exploitations of Spatial Constraints in Diagrammatic Reasoning. In Stapleton, G., Howse, J. and Lee, J. (eds.), *Diagrammatic representation and inference* (Vol. 5223, pp. 74-88): Springer Berlin / Heidelberg.
- Shin, S. J. (1994) *The Logical Status of Diagrams*, Cambridge: CUP.
- Shum, S. (1991) Cognitive Dimensions of Design Rationale. *People and Computers VI: Proceedings of HCI'91*. D. Diaper and N. V. Hammond. Cambridge, Cambridge University Press.
- Siochi, A. and Hix, D. (1991) A Study of Computer-Supported User Interface Evaluation Using Maximal Repeating Pattern Analysis. In: *ACM SIGCHI 1991*, New Orleans, LA: 301-305.
- Soller, A. and Lesgold, A. (2003) A Computational Approach to Analyzing Online Knowledge Sharing Interaction. *Artificial Intelligence in Education: Shaping the future of learning through intelligent technologies*. F. V. J. K. U. Hoppe. Amsterdam, IOS Press: 253-268.
- Stapleton, G. (2005) A Survey of Reasoning Systems Based on Euler Diagrams. *Electronic Notes in Theoretical Computer Science* 134(0): 127-151.
- Stapleton, G. and Delaney, A. (2007) Towards overcoming deficiencies in constraint diagrams In: *VLHCC '07 Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing*. IEEE Computer Society, Berlin, pp. 33-40.
- Stapleton, G. and Delaney, A. (2008) Evaluating and Generalizing Constraint Diagrams. *Journal of Visual Languages and Computing*, 19(4): 499-521.

- Stapleton, G., Howse, J. and Taylor, J. (2003) A Constraint Diagram Reasoning System. In proceedings of 9th International Conference on Distributed Multimedia Systems, Visual Languages and Computing, Florida, USA, pages 263-270, Knowledge Systems Institute.
- Stapleton, G., Howse, J. and Taylor, J. (2005a) A Decidable Constraint Diagram Reasoning System. *Journal of Logic and Computation* 15(6): 975-1008.
- Stapleton, G., Howse, J., Chapman, P., Delaney, A., Burton, J. and Oliver, I. (2013) Formalizing Concept Diagrams. 19th International Conference on Distributed Multimedia Systems, International Workshop on Visual Languages and Computing, Knowledge Systems Institute.
- Stapleton, G., Howse, J., Taylor, J. and Thompson, S. (2004a) What Can Spider Diagrams Say? *Diagrammatic Representation and Inference*. 2980: 112-127.
- Stapleton, G., Howse, J., Taylor, J. and Thompson, S. (2004b) The Expressiveness of Spider Diagrams Augmented with Constants. In: *Proceedings of the Visual Languages and Human Centric Computing*, Rome, Italy, IEEE Computer Society Press, Silver Spring, MD, pp. 91-98.
- Stapleton, G., Howse, J., Taylor, J. and Thompson, S. (2004c) The Expressiveness of Spider Diagrams. *Journal of Logic Computation* 14(6):857-880.
- Stapleton, G., J. Howse, Chapman, P., Oliver, I. and Delaney, A. (2012) What Can Concept Diagrams Say? *Diagrams* 2012: 291-293.
- Stapleton, G., Taylor, J., Thompson, S. and Howse, J. (2009) The Expressiveness of Spider Diagrams Augmented with Constants. *Journal of Visual Languages and Computing*, 20 (1):91-98.
- Stapleton, G., Thompson, S., Fish, A., Howse, J. and Taylor, J. (2005b) A New Language for the Visualization of Logic and Reasoning, In proceedings of 11th International Conference on Distributed Multimedia Systems, Visual Languages and Computing, Banff, Canada, pages 287-292, Knowledge Systems Institute.
- Stenning, K. (2002) *Seeing Reason: Image and Language in Learning to Think*, Oxford University Press.
- Stenning, K. and Lemon, O. (2001) Aligning Logical and Psychological Perspectives on Diagrammatic Reasoning. *Artificial Intelligence Review* 15(1/2): 29-62.
- Stenning, K. and Oberlander, J. (1995) A Cognitive Theory of Graphical and Linguistic Reasoning: Logic and Implementation. *Cognitive Science* 19(1): 97-140.
- Taylor, K. L. and Dionne J. P. (2000) Accessing Problem-Solving Strategy Knowledge: The Complementary Use of Concurrent Verbal Protocols and Retrospective Debriefing. *Journal of Educational Psychology* 92(3): 413-425.
- Thièvre, J., Viaud, M. and Verroust-Blondet, A. (2005) Using Euler Diagrams in Traditional Library Environments. In: *Euler Diagrams 2004*, in: ENTCS, vol. 134, pp. 189-202.

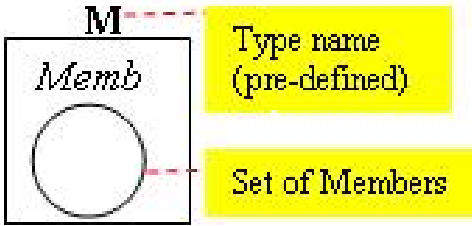


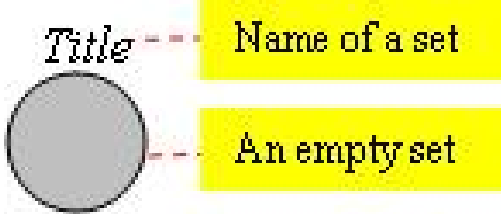
- Tufte, E. R. (1997) *Visual Explanations: Images and Quantities. Evidence and Narrative*, Cheshire, CT: Graphics Press, Cheshire, CT.
- VanLehn, K. (1989) Discovering Problem Solving Strategies: What Humans Do and Machines Don't (Yet). *Proceedings of the Sixth International Workshop on Machine Learning*. A. Segre. Los Altos, CA, Morgan Kaufmann: 215-217.
- VanLehn, K., Burleson, W., Chavez Echeangary, H., Christopherson, R., Gonzales Sanchez J., Hastings, J., Hidalgo Pontet, Y., Muldner, K. and Zhang, L. (2011) The Level Up Procedure: How to Measure Learning Gains Without Pre- and Post-testing. In Hirashima, T. et al. (eds.), *Proceedings of the 19th International Conference on Computers in Education*. Chiang Mai, Thailand: Asia-Pacific Society for Computers in Education.
- Verroust, A. and Viaud, M. L. (2004) Ensuring the Drawability of Euler Diagrams for Up to Eight Sets. In: *Proceedings of 3rd International Conference on the Theory and Application of Diagrams*, in: LNAI, vol. 2980, Springer, Cambridge, UK, pp. 128-141.
- Vicente, K. J. (1999) Ecological Interface Design: Supporting Operator Adaptation, Continuous Learning, Distributed, Collaborative Work. In *Proceedings of the Human Centered Processes Conference*, 93-97.
- Warmer, J. and Kleppe, A. (1998) *The Object Constraint Language: Precise Modeling with UML*. Addison-Wesley.
- Wharton, C., Rieman, J., Lewis, C. and Poison, P. (1994) *The Cognitive Walkthrough Method: A Practitioner's Guide*. Usability Inspection Methods. J. N. R. L. Mack. New York, John Wiley and Sons Inc.: 105-140.
- Wing, J. M. (2006) Computational Thinking. *CACM*, viewpoint 49(3): 33-35.
- Woodcock, J. and Davies, J. (1996) *Using Z: Specification, Refinement, and Proof*. Prentice-Hall.
- Yang, S., Burnett, M., Dekoven, E. and Zloof, M. (1997) Representation Design Benchmarks: A Design-Time Aid for VPL Navigable Static Representations. *Journal of Visual Languages and Computing* 8(5-6): 563-599.
- Yazdani, M. and Ford, L. (1996) Reducing the Cognitive Requirements of Visual Programming. *IEEE Symposium on Visual Languages*, Boulder, CO: 255-262.
- Zhang, J. (1996) A Representational Analysis of Relational Information Displays. *International Journal of Human Computer Studies* 45(59-74).
- Zhao, Y. and Lövdahl, J. (2003) A Reuse Based Method of Developing the Ontology for E-Procurement. In: *Proceedings of the Nordic Conference on Web Services (NCWS)*, Vaxjö, Sweden.

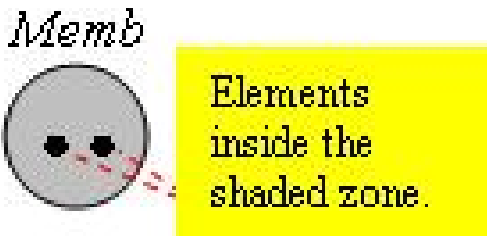



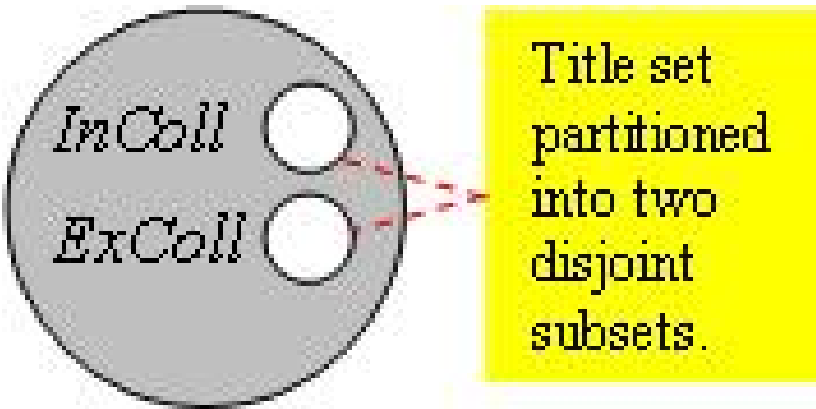


Appendix A: Material used for conducting Experiment 1

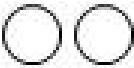
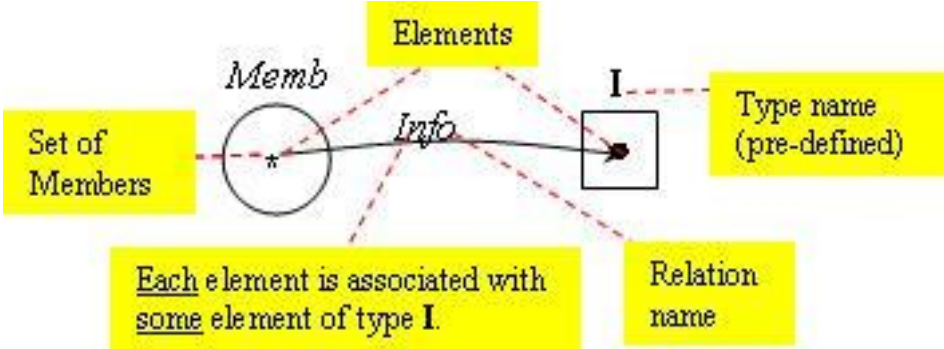
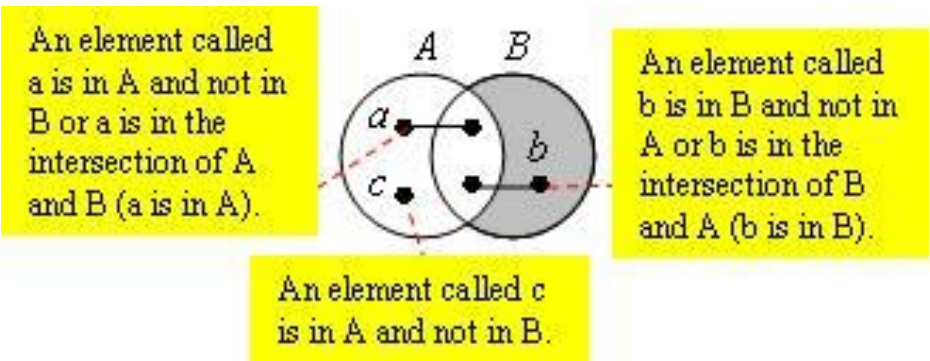
In this Appendix we will present the material that used in Experiment 1. *Italic words* are the names of things that are defined in specifying our information system and **bold words** are their type. At the beginning participants were given a training example followed by three training questions. Then each example that illustrates a particular concept will be followed by three questions. The total are eight examples each followed by three questions about that concept.

There will be three tables. Table (A.1) represents the Examples used for experiment 1 for the CD group.


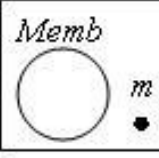
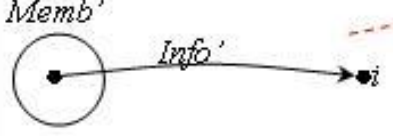
Table A.1 The Examples for the CD group

Example #	Example content for the CD group
Training Example	$X-2*4=10$ <p>This equation represents the value of x. $x-2*4=10$ means $x-8=10$ which means $x=10+8$. So $x=18$.</p>
Example 1: Sets and Types	 <p>This diagram defines Memb (Member) to be a finite subset of given type M.</p> <p>Definition:</p>  <p>This represents a set that can have a number of elements, zero or more. It may have a name.</p>  <p>This represents a predefined type. It will have a name.</p>
Example 2: Members of sets	

	<p>This diagram represents an empty set called <i>Title</i>.</p> <div data-bbox="518 257 1007 492">  </div> <p>This diagram represents a set called <i>Memb</i>. With two elements only.</p> <p>Definition:</p> <div data-bbox="512 663 1449 1039">  <p>This shaded zone shows that the set does not contain any elements other than the shown ones. This zone has no elements.</p>  <p>An element.</p>  <p>Elements inside this shaded zone show the set has a number of elements. This zone has a number of elements less than or equal to three.</p> </div>
<p>Example 3: Sets' relationships</p>	<div data-bbox="518 1097 1337 1503">  </div> <p>This diagram divides <i>Title</i> into two disjoint subsets: those having no copies are said to be <i>ExColl</i> or 'Ex-Collection'; the others are said to be <i>InColl</i> or 'In-Collection'.</p> <p>Definition: There are three types of relationships between sets</p> <div data-bbox="493 1720 1449 1968">  <p>Two sets can intersect: A set can have some members that are already members of another set.</p>  <p>Two sets can overlap: A set can be a subset of another set.</p> </div>

	 <p>Two sets can be disjoint: Two sets can be partitioned into separate subsets.</p>
Example 4: Relationships	 <p>This diagram defines a relationship called <i>Info</i> that associates every element of <i>Memb</i> with some element of type I.</p> <p>Definition:</p> <ul style="list-style-type: none"> * This represents all elements in a specific set. ● This represents the existence of an element. → This represents a relationship between two sets, elements or a combination of both.
Example 5: Spiders	 <p>An element called <i>a</i> is in A and not in B or <i>a</i> is in the intersection of A and B (<i>a</i> is in A).</p> <p>An element called <i>b</i> is in B and not in A or <i>b</i> is in the intersection of B and A (<i>b</i> is in B).</p> <p>An element called <i>c</i> is in A and not in B.</p> <p>This diagram defines two sets A and B that intersect. There exist at least three elements; <i>a</i>, <i>b</i> and <i>c</i>.</p> <p>Definition:</p> <ul style="list-style-type: none"> ● — ● Elements in different zones are connected by straight lines to denote a spider. For a spider, elements are called feet and straight lines are called legs. A spider denotes the existence of an element.

<p>Example 6: Spiders' relationships</p>	<div data-bbox="539 255 1394 501"> </div> <p>This diagram defines two intersected sets A and B. There is an element in A and an element in B which may or may not be the same element.</p> <p>Definition:</p> <div data-bbox="507 698 592 730"> </div> <p>A strand connects two feet, from different spiders, placed in the same zone . The two spiders may represent the same element.</p>
<p>Example 7: Invariant</p>	<div data-bbox="507 871 1270 1509"> </div> <p>A diagram introduced shows that a Video Copy class (<i>VC</i>) maintains a set of <i>Title</i> (s) (uniquely identified by elements from given type T). Each known <i>Title</i> has their own associated <i>Desc</i> (Description) (of type D), and is either in <i>InColl</i> (In-Collection) and in <i>ExColl</i> (Ex-Collection), but not in both.</p> <p>Definition:</p> <p>The general form is:</p>

	<div style="text-align: center;"> <div style="border: 1px solid black; padding: 5px; display: inline-block;">Core Concept name</div> <div style="display: flex; align-items: center; justify-content: center;"> <div style="border: 1px solid black; padding: 5px; margin-right: 10px;">Semi-box.</div> <div style="border: 1px solid black; padding: 10px; text-align: center;"> C $STATE-INVARIANT$ </div> </div> </div> <div style="border: 1px solid black; padding: 5px; margin-top: 10px; text-align: center;"> Statements of the core concept go here. They are the conditions that are always true about the class. </div> <p>This framework represents a core concept (class). Its state-invariant is written as any mix of declarations and predicates (separated by a ';' when they appear on the same line). State-invariants of the core concept are the conditions that are always true about the class.</p>
<p>Example 8: An Event Specification</p>	<div style="text-align: center;"> $VM!NewMember(m,i)$ <div style="display: flex; justify-content: space-around; align-items: flex-start;"> <div style="border: 1px solid black; padding: 10px; text-align: center;"> I  </div> <div style="border: 1px solid black; padding: 10px; text-align: center;"> M  </div> </div> <div style="border: 1px solid black; padding: 10px; margin-top: 10px;">  </div> </div> <div style="border: 1px solid black; padding: 5px; margin-top: 10px; text-align: center;"> The pre-condition ensures that any parameter values satisfy all constraints which are imposed by the state-invariant. </div> <div style="border: 1px solid black; padding: 5px; margin-top: 10px; text-align: center;"> In the post-condition, dashed names denote values that are changed- but minimal changes are shown here; there is no need to say that other elements of <i>Memb</i> and their associated <i>Info</i> values remain the same, because of the convention that "the rest stays unchanged". </div> <p>An event (operation) called $VM!NewMember(m,i)$ can be used to register a new member, m, with information i. The pre-condition (before applying the event) ensures that i has type I, and that m is an identifier of type M which is not in <i>Memb</i>. In the post-condition(after applying that even), dashed names denote values that are changed - but only minimal changes are shown here; there is no need to say that other elements of <i>Memb</i> and their associated <i>Info</i> values remain the same, because of the convention that "the rest stays unchanged"</p> <p>Definition:</p> <p>The general form is:</p> <div style="text-align: center; margin-top: 20px;"> <div style="border: 1px solid black; padding: 5px; display: inline-block;">Event Separator</div> <div style="display: flex; align-items: center; justify-content: center;"> <div style="border: 1px solid black; padding: 10px; margin-right: 10px;"> $C!E(... ..)$ $PRE-CONDITION$ <hr/> $POST-CONDITION$ </div> </div> </div> <div style="border: 1px solid black; padding: 5px; margin-top: 10px; text-align: center;"> The possible states after applying that event. </div> <div style="border: 1px solid black; padding: 5px; margin-top: 10px; text-align: center;"> The allowable states for applying that event. </div> <p>This form represents a named event $C!E$ which specifies some allowable change-of-state</p>

	for such objects. Thus it always involves a POST-CONDITION (below the double line), and it may include an optional PRE-CONDITION (above) as well. An event may have input-arguments (names between parentheses); these denote local constants, which must be declared in its pre-condition.
--	---

The next table (A.2) represents the Examples used for experiment 1 for the NL group.

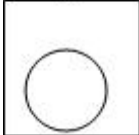
Table A.2 The Examples for the NL group

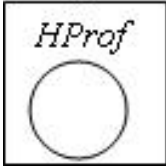
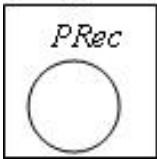

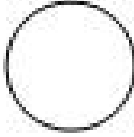
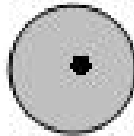
Example #	Example content for the NL group
Training Example	<div>The product of 2 and 4 subtracted from x equals 10.</div> <p>This statement represents the value of x. $x - 2 * 4 = 10$ means $x - 8 = 10$ which means $x = 10 + 8$. So, $x = 18$.</p>
Example 1: Sets and Types	<div>There is a set called <i>Member</i> of type M.</div> <p>Definition:</p> <p>A set can have a number of elements, zero or more. It may have a name.</p> <p>We may say that a set can be of a particular type (have a particular property).</p>
Example 2: Members of sets	<div>There is a set called <i>Title</i> which is an empty set of elements.</div> <div>There is a set called <i>Member</i> which has a fixed number of</div> <p>Definition:</p> <ul style="list-style-type: none"> - A set with zero number of elements is called an empty set. - Any member of a set is called an element. - A set can have a fixed number of elements.
Example 3: Sets' relationships	<div><i>Title</i> set is partitioned into two separated subsets: those who have no copies are said to be <i>Ex-Collection</i>; all others are said to be <i>In-Collection</i>.</div> <p>Definition:</p> <p>There are three types of relation between sets:</p> <ol style="list-style-type: none"> 1. A set can have some members that are already members of another set. 2. A set can be a subset of another set. 3. Two sets can be partitioned into separated subsets.
Example 4: Relationships	<div>Every element of a set called <i>Member</i> of type M is associated with some element of type I by a relation called <i>Information</i>.</div> <p>Definition:</p>



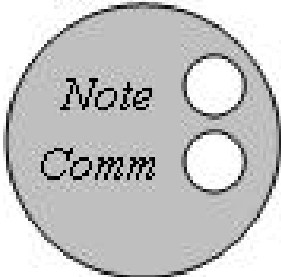
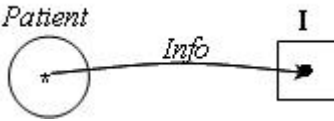
	Two sets, elements, or a combination of both can relate to each other by a relation that defines the association
Example 5: Spiders	<div> <p>Consider two sets A and B that intersect and three elements; a, b and c.</p> <p>An element a is in A and not in B or a is in the intersection of A and B (a is in A).</p> <p>An element b is in B and not in A or b is in the intersection of B and A (b is in B).</p> <p>An element c is in A and not in B.</p> </div> <p>Definition:</p> <p>Elements in the same location <u>are the same</u> in a given situation if they denote the existence of an element</p>
Example 6: Elements' relationships	<div> <p>Consider two intersected sets A and B and two elements. There is an element in A and an element in B which may or may not be the same element.</p> </div> <p>Definition:</p> <p>Two different elements in the <u>same location</u> may be the same in a given situation.</p>
Example 7: Invariant	<div> <p>A <i>Video Copy</i> class maintains a set of <i>Title</i> (s) (uniquely identified by elements from given type T). Each known <i>Title</i> has its own associated <i>Description</i> (of type D), and is either in <i>in-collection</i> and in <i>ex-collection</i>, but not in both.</p> </div> <p>Definition:</p> <p>A core concept (class) has several components; a class name and a state-invariant which is written as any mix of statements of declarations and predicates. State-invariants of the core concept are the conditions that are always true about the class.</p>
Example 8: An Event Specification	<div> <p>An event (operation) called <i>NewMember(m,i)</i> is used to register a new <i>Member</i>, <i>m</i>, with Information, <i>i</i>. The pre-condition (before applying the event) ensures that its input-argument <i>m</i> is an identifier of type M which is not in <i>Member</i>. And the other input-argument <i>i</i> has type I. In the post-condition (after applying that event), values are changed. <i>m</i> is associated with <i>i</i> by a relationship called <i>Information</i> and it is in <i>Member</i>.</p> </div> <p>Definition:</p> <p>An event specifies some allowable change-of-state for such objects. Thus it always involves a POST-CONDITION and it may include an optional PRE-CONDITION as well. An event may have input-arguments (names between parentheses); these denote local constants, which must be declared in its pre-condition. The pre-condition ensures that any parameter values satisfy all constraints which are imposed by the state-invariant.</p>


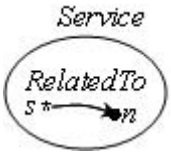

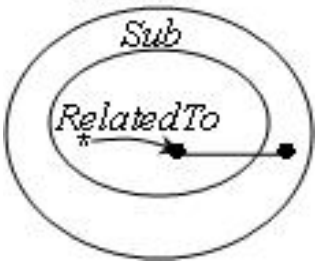
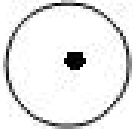
After each example three questions are presented to the participants. For each question the answer will be “Yes”, “No” or “Not Specified”. “Not Specified” means there is insufficient information provided in the question to answer the question without any assumption. Before answering, participants could return to the related example at any time. However, after answering they only could rate their confidence about their performance by choosing the level of how hard the question was. There were five levels: very difficult, difficult, intermediate, easy, and very easy. After rating, an immediate feedback about the result was shown to the participant to tell them if their answer was right or wrong. If the answer was wrong they would be provided with the correct answer because this experiment was a training experiment. Table A.3 represents the 24 questions for both groups along with the answers.


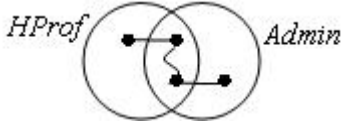
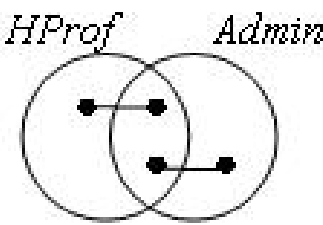
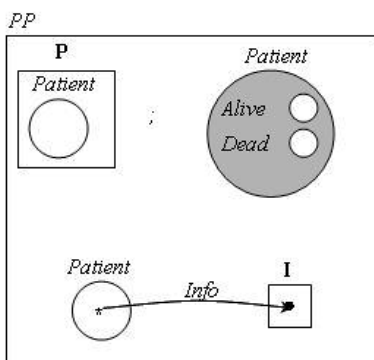
Table A.3 The Questions for both the CD and the NL groups and the correct answer

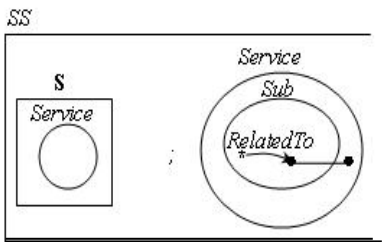
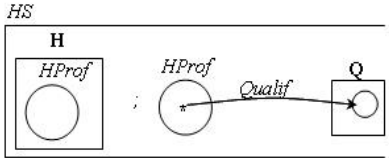
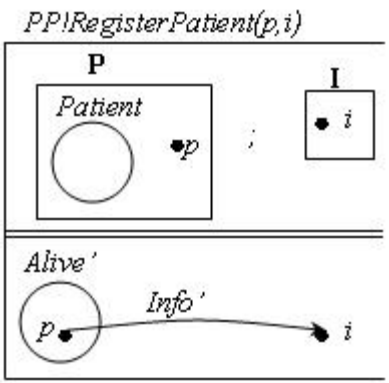
	CD	NL	The answer
Training Q1	$30+x/10=40$ <p>From the given equation, is it true that $x = 100$?</p>	<div style="border: 1px solid black; padding: 5px; margin-bottom: 5px;"> x divided by 10 added to 30 equals 40 </div> <p>From the given statement, is it true that $x = 100$?</p>	Yes
Training Q2	$z+x/10=40$ <p>From the given equation, is it true that $z = 50$?</p>	<div style="border: 1px solid black; padding: 5px; margin-bottom: 5px;"> x divided by 10 added to z equals to 40. </div> <p>From the given statement, is it true that $z = 50$?</p>	Not Specified
Training Q3	$10+y=40$ <p>From the given equation, is it true that $y = 10$?</p>	<div style="border: 1px solid black; padding: 5px; margin-bottom: 5px;"> 10 added to y equals to 40. </div> <p>From the given statement, is it true that $y = 10$?</p>	No
Q1	<p style="text-align: center;">P</p>  <p>From the given diagram, is there a set of patients called <i>Patient</i> of type P?</p>	<div style="border: 1px solid black; padding: 5px; margin-bottom: 5px;"> A set is uniquely identified by elements from given type P. </div> <p>From the given statement, is there a set of patients called <i>Patient</i> of type P?</p>	Not Specified

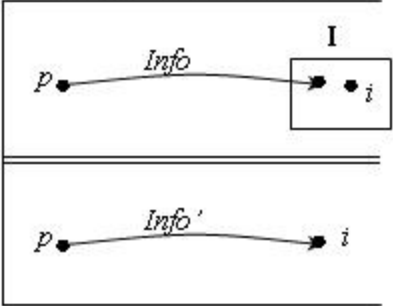
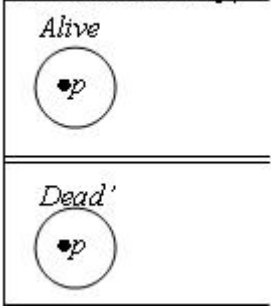
Q2	<p style="text-align: center;">H</p>  <p>From the given diagram, is there a set called <i>Health-Professional</i> of type H?</p>	<div style="border: 1px solid black; padding: 5px; margin-bottom: 10px;"> <p><i>Health-Professionals</i> are uniquely identified by elements from type H.</p> </div> <p>From the given statement, is there a set called <i>Health-Professional</i> of type H?</p>	Yes
Q3	<p style="text-align: center;">R</p>  <p>From the given diagram, is R a set of type <i>PRec</i>?</p>	<div style="border: 1px solid black; padding: 5px; margin-bottom: 10px;"> <p><i>Patient-Records</i> are uniquely identified by elements from type R.</p> </div> <p>From the given statement, is R a set of type <i>Patient Record</i>?</p>	No
Q4	<p style="text-align: center;"><i>Patient</i></p>  <p>From the given diagram, is it true that there are no patients in <i>Patient</i> set?</p>	<div style="border: 1px solid black; padding: 5px; margin-bottom: 10px;"> <p>There is a set called <i>Patient</i> and it is an empty set.</p> </div> <p>From the given statement, is it true that there are no patients in <i>Patient</i> set?</p>	Yes
Q5	<p style="text-align: center;"><i>Patient</i></p>  <p>From the given diagram, is it true that there are many elements in <i>Patient</i> set?</p>	<div style="border: 1px solid black; padding: 5px; margin-bottom: 10px;"> <p>There could be a set, called <i>Patient</i>, of patients.</p> </div> <p>From the given statement, is it true that there are many elements in <i>Patient</i> set?</p>	Not Specified
Q6	<p style="text-align: center;"><i>Patient</i></p>  <p>From the given diagram, is it true that there are no patients in <i>Patient</i> set?</p>	<div style="border: 1px solid black; padding: 5px; margin-bottom: 10px;"> <p>There is a set called <i>Patient</i> with only one element.</p> </div> <p>From the given statement, is it true that there are no patients in <i>Patient</i> set?</p>	No

Q7	 <p>From the given diagram, can an element of <i>HProf</i> be both a <i>Doctor</i> and a <i>Nurse</i>?</p>	<p><i>Health-Professional</i> set is partitioned into two disjoint subsets; those who are <i>Doctors</i>; all others are said to be <i>Nurses</i>.</p> <p>From the given statement, can an element of <i>Health-Professional</i> be both a <i>Doctor</i> and a <i>Nurse</i>?</p>	No
Q8	 <p>From the given diagram, can an element of <i>Patient</i> be a child?</p>	<p><i>Patient</i> set is partitioned into two separated subsets; those who are dead are said to be <i>Dead</i>; all others are said to be <i>Alive</i>.</p> <p>From the given statement, can an element of <i>Patient</i> be a child?</p>	Not Specified
Q9	 <p>From the given diagram, can an element of <i>PRec</i> be either a <i>Note</i> or a <i>Comm</i>?</p>	<p><i>Patient-Record</i> set is partitioned into two separated subsets; those who are said to be <i>Note</i>; all others are said to be <i>Communication</i>.</p> <p>From the given statement, can an element of <i>Patient-Record</i> be either a <i>Note</i> or a <i>Communication</i>?</p>	Yes
Q10	 <p>From the given diagram, can two elements of <i>Patient</i> be associated with the same piece of <i>Info</i>?</p>	<p>Each element of <i>Patient</i> is associated with some <i>Information</i> of type <i>I</i>.</p> <p>From the given statement, can two elements of <i>Patient</i> be associated with the same piece of <i>Information</i>?</p>	No

Q11	 <p>From the given diagram, do all <i>HProf</i> have the same <i>Qualif</i>?</p>	<p>Each element of <i>Health-Professional</i> is associated with a set of <i>Qualification</i> of type <i>Q</i>.</p> <p>From the given statement, do all <i>Health-Professional</i> have the same <i>Qualification(s)</i>?</p>	Not Specified
Q12	 <p>From the given diagram, must <i>s</i> be <i>RelatedTo n</i>?</p>	<p>A <i>Service s</i> can be a <i>Sub-service</i> of a service <i>n</i>.</p> <p>From the given statement, must <i>s</i> be <i>RelatedTo n</i>?</p>	Yes
Q13	 <p>From the given diagram, are there two services?</p>	<p>A set called <i>Service</i> has two elements.</p> <p>From the given statement, are there two services?</p>	Yes
Q14	 <p>From the given diagram, must every <i>Service</i> have <i>Sub-service</i>?</p>	<p>A set called <i>Service</i> has a sub-set called <i>Sub</i>. All <i>Sub</i>'s elements are related to other elements which may be in <i>Sub</i> and eventually must be related to some element in <i>Service</i>.</p> <p>From the given statement, must every <i>Service</i> have <i>Sub-service</i>?</p>	No
Q15	 <p>From the given diagram, is there only one service?</p>	<p>A set called <i>Service</i> can have one element.</p> <p>From the given statement, is there only one service?</p>	Not Specified

Q16	 <p>From the given diagram, are there at most two services?</p>	<p>A set called <i>Service</i> may have two elements or less.</p> <p>From the given statement, are there two services at most?</p>	Yes
Q17	 <p>From the given diagram, it is true that a <i>HProf</i> must be an <i>Admin</i>?</p>	<p>If a <i>Health-Professional</i> works as an <i>Administrator</i> and an <i>Administrator</i> works as a <i>Health-Professional</i>, then these both may be the same person.</p> <p>From the given statement, it is true that a <i>Health-Professional</i> must be an <i>Administrator</i>?</p>	No
Q18	 <p>From the given diagram, should at most one <i>HProf</i> be an <i>Admin</i>?</p>	<p>If a <i>Health-Professional</i> works as an <i>Administrator</i> and an <i>Administrator</i> works as a <i>Health-Professional</i>, then these both may be the same person.</p> <p>From the given statement, should at most one <i>Health-Professional</i> be an <i>Administrator</i>?</p>	Not Specified
Q19	 <p>From the given diagram, is this concept about <i>Patient</i> definition, <i>Info</i>, and gender?</p>	<p>A <i>Patient Population</i> class maintains a set of <i>Patient</i> (s) (uniquely identified by elements from given type P). Each known <i>Patient</i> has their own associated Information (of type I), and is either <i>Alive</i> or in the end <i>Dead</i>.</p> <p>From the given statement, is this concept about <i>Patient</i> definition, <i>Information</i>, and gender?</p>	Not Specified

Q20	 <p>From the given diagram, is this concept about defining services and subservices only?</p>	<p>A <i>Service System</i> class is the concept which has conditions related to some hierarchical set of <i>Service</i> (s) and nested <i>Sub-services</i> (uniquely identified as an element of type S).</p> <p>From the given statement, is this concept about defining services and subservices only?</p>	Yes
Q21	 <p>From the given diagram, is this concept about <i>Medical Equipment</i> ?</p>	<p>A <i>Health System</i> class is the concept which has conditions related to <i>Health-Professional</i> (uniquely identified as an element of type H). Each <i>Health-Professional</i> has a set of <i>Qualification</i> (s) (uniquely identified as a set of type Q).</p> <p>From the given statement, is this concept about <i>Medical Equipment</i>?</p>	No
Q22	 <p>From the given diagram, can an element of <i>Patient</i> be re-registered?</p>	<p>An event (operation) called '<i>RegisterPatient(p,i)</i>' can be used to register a new patient, <i>p</i>, with information <i>i</i>; initially, <i>p</i> is said to be <i>Alive</i>. <u>Before performing this event <i>i</i> has to be of type I, and <i>p</i> is an identifier of type P which is not in <i>Patient</i>. After performing this event <i>p</i> will be in <i>Patient</i> and <i>p</i> will be associated with I by the relation <i>Information</i>.</u></p> <p>From the given statement, can an element of <i>Patient</i> be re-registered?</p>	No


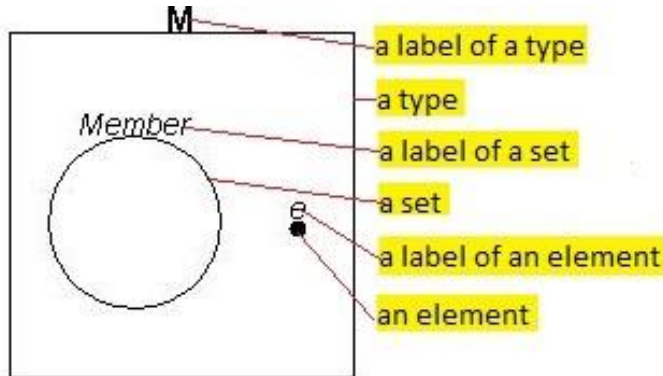



Q23	<p style="text-align: center;"><i>PP!UpdatePatientInfo(p,i)</i></p>  <p>From the given diagram, can <i>Info</i> be updated?</p>	<div style="border: 1px solid black; padding: 10px; margin-bottom: 10px;"> <p>An event, called '<i>UpdatePatientInformation(p,i)</i>', can be used to update current information for a patient, <i>p</i>, so it has some value <i>i</i> afterwards. <u>Before performing this event</u> <i>p</i> should be an element of <i>Patient</i> and <i>i</i> should be a value of type <i>I</i> which differs from the current value of <i>Information</i>; so <u>After performing this event</u> that value will indeed be changed, whilst still preserving the invariant of <i>PP</i>.</p> </div> <p>From the given statement, can <i>Information</i> be updated?</p>	Yes
Q24	<p style="text-align: center;"><i>PP!RecordDeath(p)</i></p>  <p>From the given diagram, has a male <i>Patient</i> died?</p>	<div style="border: 1px solid black; padding: 10px; margin-bottom: 10px;"> <p>An event, called '<i>RecordDeath(p)</i>', can be used to record the death of patient <i>p</i>, but only if <i>p</i> is <i>Alive</i>. <u>Before performing this event</u> <i>p</i> should be an element of <i>Alive</i> initially which is a partition of <i>Patient</i>; so <u>after performing this event</u> <i>p</i> will be removed from <i>Alive</i> to <i>Dead</i>, whilst still preserving the invariant of <i>PP</i>.</p> </div> <p>From the given statement, has a male <i>Patient</i> died?</p>	Not Specified

Appendix B: Material used for conducting Experiment 2

In this Appendix we will present the material that used in Experiment 2. *Italic* words are the names of things that are defined in specifying our information system and **bold** words are their type. At the beginning participants were given a training example followed by three training questions. Then each example that illustrates a particular concept will be followed by three questions. The total are seven examples each followed by three questions about that concept.

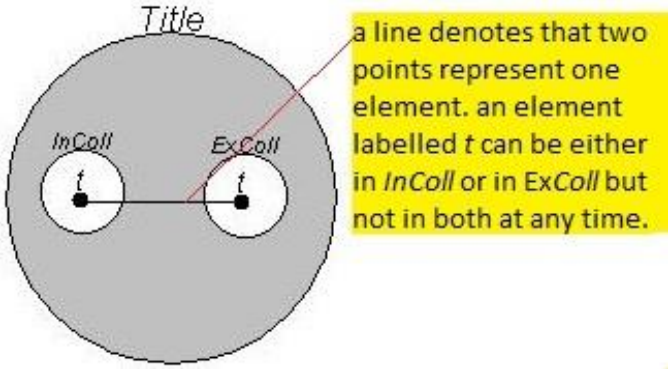
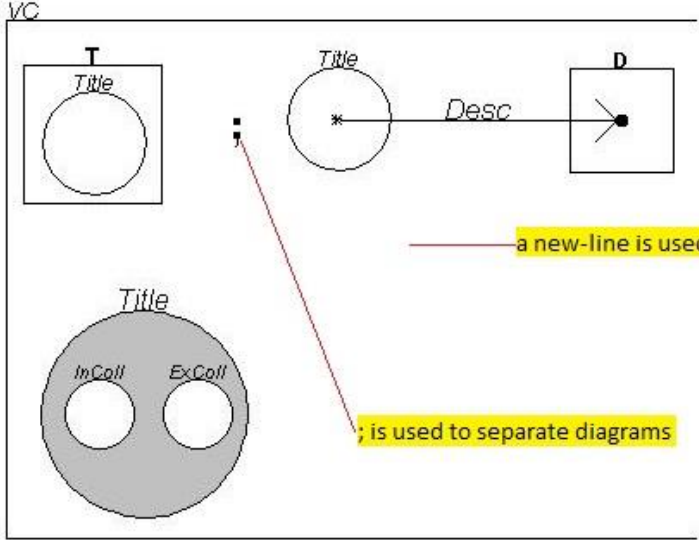
There will be three tables. Table (B.1) represents the Examples used for experiment 2 for the CD group. The diagrams here were drawn by using our developed editor.

Table B.1 The Examples for the CD group

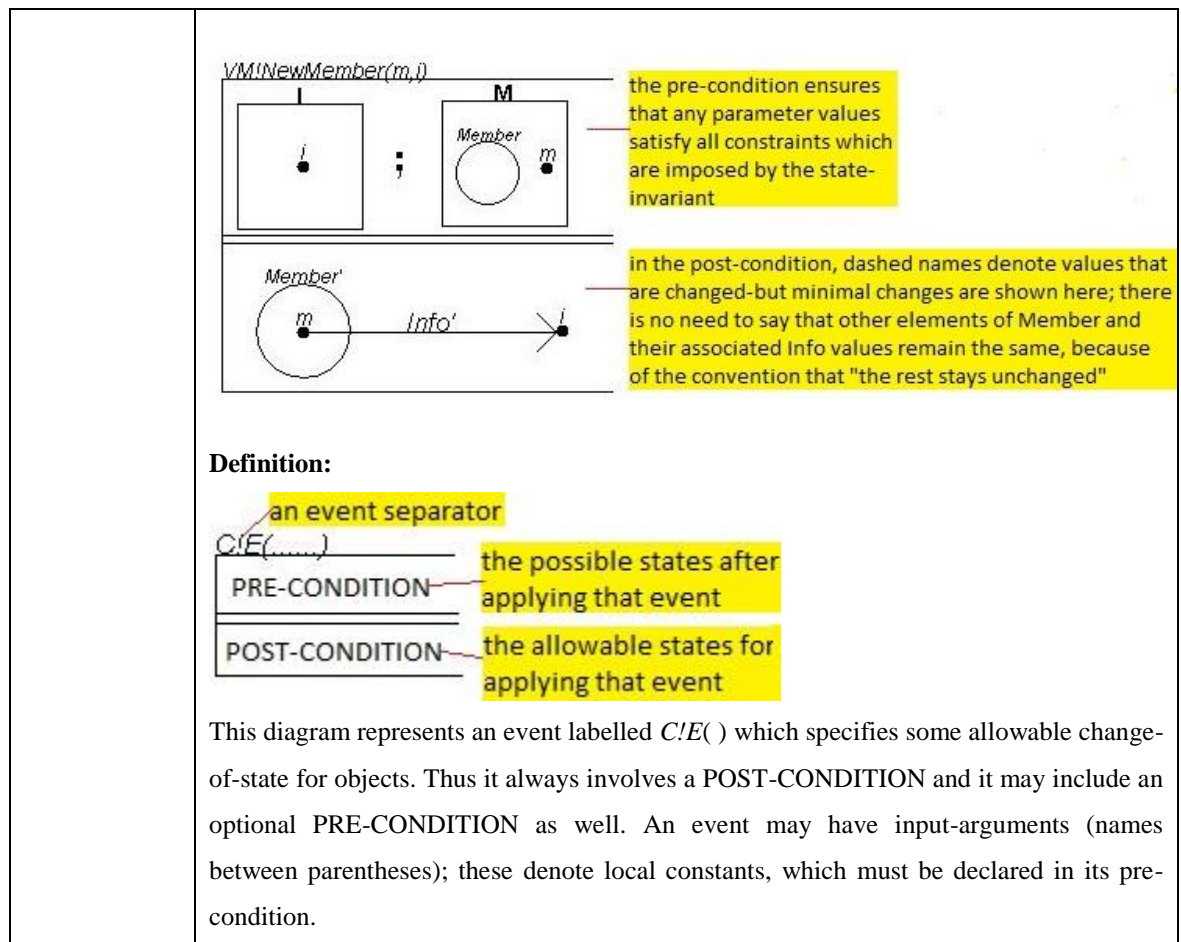
Example #	Example content for the CD group
Training Example	<p>There is a set of elements. This set is labelled <i>A</i>.</p> 
Example 1: Types Sets and Elements	<p>There is a set labeled <i>Member</i>. The elements of that set shared a particular type labeled M. Also, there is an element labelled <i>e</i> of type M. However, <i>e</i> is not in the set <i>Member</i>. The following diagram represents this statement:</p>  <p>Definition:</p> <p> This diagram represents a type (a category) which is a particular property of an element. Elements of a set are of a particular type. A type will have a label.</p> <p> This diagram represents a set. A set has zero or more elements. It may have a label.</p> <p> This diagram represents an element. It may have a label.</p>

<p>Example 2: Members of Sets</p>	<p>There is a set labeled <i>Member</i> with at least two elements. This means that there are two members or more in the set <i>Member</i>.</p> <p>The following diagram represents this statement.</p> <p>There is an empty set labelled <i>Title</i> which has no elements. This means that there are no titles in the set <i>Title</i>.</p> <p>The following diagram represents this statement.</p> <p>There is a set labeled <i>Member</i> with exactly two elements. This means that there are exactly two members in the set <i>Member</i>.</p> <p>The following diagram represents this statement.</p> <p>Definition:</p> <p>This diagram represents a set which may contain elements more than the shown ones. This set has a number of elements greater than or equal three.</p> <p>This diagram represents a set which does not contain any elements other than the shown ones. This set has no elements. A set with zero number of elements is defined as an empty set.</p> <p>This diagram represents a set which does not contain any elements other than the shown ones. This set has exactly three elements.</p>
<p>Example 3: Sets Relations</p>	<p>There is a set labeled <i>Title</i> which is divided into two disjoint subsets: those having no copies are labelled as <i>ExColl</i> or '<i>Ex-Collection</i>'; the others are labelled as <i>InColl</i> or '<i>In-Collection</i>'.</p> <p>The following diagram represents this statement.</p>

	<div data-bbox="507 241 1018 609" data-label="Diagram"> </div> <p>Definition:</p> <p>There are three types of relations between sets:</p> <div data-bbox="512 786 655 875" data-label="Diagram"> </div> <p>This diagram represents two sets which intersect; a set can have some elements that are also elements of another set.</p> <div data-bbox="512 943 639 1066" data-label="Diagram"> </div> <p>This diagram represents a set which contain another set; a set can be a subset of another set. All the elements in the subset are in the set.</p> <div data-bbox="512 1144 699 1234" data-label="Diagram"> </div> <p>This diagram represents two sets which be disjoint; they have no elements in common.</p>
<p>Example 4: Relations</p>	<p>There is a relation labelled <i>Info</i> (Information) that associates every element of <i>Member</i> with some element of type <i>I</i>.</p> <div data-bbox="488 1402 1437 1693" data-label="Diagram"> </div> <p>Definition:</p> <ul style="list-style-type: none"> * This diagram represents all elements in a specific set. • This diagram represents the existence of an element. → This diagram represents a relationship between two sets, elements or a combination of both.

<p>Example 5: Spiders (Possible Locations of an Element)</p>	<p>There is a title labelled <i>t</i>. This title's state can be changed from <i>ExColl</i> to <i>InColl</i> or from <i>InColl</i> to <i>ExColl</i>. In other words, at any time it is the case that <i>t</i> will be either in <i>InColl</i> or in <i>ExColl</i>, but not in both.</p> <p>The following diagram represents this statement.</p>  <p>a line denotes that two points represent one element. an element labelled <i>t</i> can be either in <i>InColl</i> or in <i>ExColl</i> but not in both at any time.</p> <p>Definition:</p> <p>— This diagram represents a straight line (called leg) which is used to denote that an element (called a spider) can be in many possible locations (called feet).</p>
<p>Example 6: Invariant</p>	<p>There is an invariant for a <i>Video Copy</i> class and it is labelled <i>VC</i>. It is a specification that maintains a set of titles labelled <i>Title</i>. This set is uniquely identified by elements from a given type T. All titles must have their own associated descriptions of type D. This association is labeled <i>Desc</i>. A title is either in <i>InColl</i> (In-Collection) or in <i>ExColl</i> (Ex-Collection), but not in both.</p> <p>The following diagram represents this statement.</p>  <p>a new-line is used to separate diagrams</p> <p>; is used to separate diagrams</p> <p>Definition:</p>

	<div data-bbox="491 264 1056 510" data-label="Diagram"> </div> <p>This diagram represents an invariant for a class. The invariant has two components; a class name and a state-invariant. The state-invariant is a mix of diagrams (separated by a ‘;’ when they appear on the same line). These diagrams declare and predicate the conditions which must be always true about this class.</p>
<p>Example 7: An Event Specification</p>	<p>There is an invariant labelled <i>VM</i>. his invariant has the definition of <i>Member</i> set and the <i>Info</i> relation as shown below:</p> <div data-bbox="491 891 1082 1131" data-label="Diagram"> </div> <p>Based on this invariant there is an event (operation) labeled <i>VM!NewMember(m,i)</i>. The invariant name, <i>VM</i>, which appears in the event’s name, is a reference to hidden information that previously defined. This event is used to register a new member, <i>m</i>, with information <i>i</i>. A specification of an event has 2 conditions; one condition (called pre-condition) is used to show things before changes such as initiated members of sets and the other condition (called post-condition) is used to show what has changed. The pre-condition (before applying the event) ensures that its input-argument, <i>i</i>, has type I, and that the other input-argument, <i>m</i>, is an identifier of type M which is not in <i>Member</i>. In the post-condition (after applying that event), <i>m</i> is associated with <i>i</i> by a relation called <i>Info</i> (Information) and it is in <i>Member</i>. Dashed names denote values that are changed- but only minimal changes are shown here; there is no need to say that other elements of <i>Member</i> and their associated <i>Info</i> values remain the same, because of the convention that “the rest stays unchanged”.</p> <p>The following diagram represents this statement.</p>



The next table (B.2) represents the Examples used for experiment 2 for the NL group.

Table B.2 The Examples for the NL group

Example #	Example content for the NL group
Training Example	There is a set of elements. This set is labelled A . Is-Set(A)
Example 1: Types Sets and Elements	There is a set called Member . The elements of that set shared a particular type called M . Also, there is an element called e of type M . However, e is not in the set Member . The following formal expression represents this statement: Is-Type(M) Is-Set(Member) Is-Element(e) Of-Type(Member,M) Of-Type(e,M)

	<p>Definition:</p> <p>Is-Type(Z) This expression represents a type (a category) called Z. A type is a particular property of an element.</p> <p>Is-Set(X) This expression represents a set called X. A set has zero or more elements.</p> <p>Is-Element(x) This expression represents an element called x.</p> <p>Of-Type(X,Z) This expression represents an element called X or elements of a set called X. X is of type Z.</p>
<p>Example 2:</p> <p>Members of Sets</p>	<p>There is a set called <i>Member</i> with at least two elements <i>a</i> and <i>b</i>. This means that there are two members or more in the set <i>Member</i>.</p> <p>The following formal expression represents this statement.</p> <p>Is-Set(Member) Is-Element(a) Is-Element(b) Element-in-Set(a,Member) Element-in-Set(b,Member) No.of-Elements(Member, \geq, 2)</p> <p>There is an empty set called <i>Title</i> which has no elements. This means that there are no titles in the set <i>Title</i>.</p> <p>The following formal expression represents this statement.</p> <p>Is-Set>Title) No.of-Elements>Title, =, 0)</p> <p>There is a set called <i>Member</i> with exactly two elements. This means that there are exactly two members in the set <i>Member</i>.</p> <p>The following formal expression represents this statement.</p> <p>Is-Set(Member) No.of-Elements(Member, =, 2)</p> <p>Definition:</p> <p>Element-in-Set(ElementName,SetName) This expression represents an element called <i>ElementName</i> which exists in a set called <i>SetName</i>. This means that this element is a member of that set.</p> <p>No.of-Elements(SetName,Operator,ElementsNumber) This expression represents a set called <i>SetName</i> which has a number of elements. The number of elements will depend on the <i>Operator</i>. If the operator is =, then this means this set has a number of elements equals <i>ElementsNumber</i>. If the <i>Operator</i> is \geq, then this means this set has a number of elements greater than the specified number.</p>

<p>Example 3: Sets Relations</p>	<p>There is a set called <i>Title</i> which is divided into two disjointed subsets: those having no copies are called as <i>ExColl</i> or '<i>Ex-Collection</i>'; the others are called as <i>InColl</i> or '<i>In-Collection</i>'.</p> <p>The following formal expression represents this statement.</p> <pre> Is-Set(Title) Is-Set(InColl) Is-Set(ExColl) Subset(InColl,Title) Subset(ExColl,Title) Disjoint(InColl,ExColl) No.of-Elements(Title,=,0) No.of-Elements(InColl,≥,0) No.of-Elements(ExColl,≥,0) </pre> <p>Definition:</p> <p>There are three types of relations between sets:</p> <p><i>Intersect(X,Y)</i> This expression represents two sets which intersect; a set called X can have some elements that are also elements of another set called Y.</p> <p><i>Subset(X,Y)</i> This expression represents a set which contain another set; a set called X is a subset of another set called Y.</p> <p><i>Disjoint(X,Y)</i> This expression represents a set called X which has no relation with another set called Y. If no relation is specified between sets, then the disjoint relation is the default.</p>
<p>Example 4: Relations</p>	<p>There is a relation called <i>Info</i> (Information) that associates every element of <i>Member</i> with some element of type I.</p> <pre> Is-Set(Member) Is-Type(I) No.of-Elements(Member,≥,0) All-Elements(m) Element-in-Set(m,Member) Some-Elements(i) Of-Type(i,I) Is-Relation(Info:m,i) </pre> <p>Definition:</p> <p><i>All-Elements(x)</i> This expression represents all elements in a set. These elements are called x.</p> <p><i>Some-Elements(y)</i> This expression represents some elements in a set or a type. These elements are called y.</p> <p><i>Is-Relation(RelationName:SourceName,DestinationName)</i> This expression represents a relation called <i>RelationName</i> which is used to represent a relation between a Source and a Destination. They could be two sets, elements or a combination of both.</p>

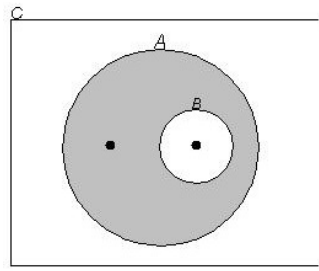
<p>Example 5: Spiders (Possible Locations of an Element)</p>	<p>There is a title called <i>t</i>. This title's state can be changed from <i>ExColl</i> to <i>InColl</i> or from <i>InColl</i> to <i>ExColl</i>. In other words, at any time it is the case that <i>t</i> will be either in <i>InColl</i> or in <i>ExColl</i>, but not in both.</p> <p>The following formal expression represents this statement.</p> <pre> Is-Set(Title) Is-Set(InColl) Is-Set(ExColl) Subset(InColl,Title) Subset(ExColl,Title) Disjoint(InColl,ExColl) No.of-Elements(Title,=,0) No.of-Elements(InColl,≥,0) No.of-Elements(ExColl,≥,0) Is-Element(t) Element-in-Set(t,InColl) Element-in-Set(t,ExColl) Same-Element(InColl.t,ExColl.t) </pre> <p>Definition:</p> <p>Same-Element(a,b) This formal expression represents that elements such as <i>a</i> and <i>b</i> in different sets denote one element.</p>
<p>Example 6: Invariant</p>	<p>There is an invariant for a <i>Video Copy</i> class and it is called <i>VC</i>. It is a specification that maintains a set of titles called <i>Title</i>. This set is uniquely identified by elements from a given type T. All titles must have their own associated descriptions of type D. This association is called <i>Desc</i>. A title is either in <i>InColl</i> (In-Collection) or in <i>ExColl</i> (Ex-Collection), but not in both.</p> <p>The following formal expression represents this statement.</p> <pre> Is-Invariant(VC) Is-Type(T) Is-Set(Title) Of-Type(Title,T) All-Elements(t) Element-in-Set(t,Title) Is-Type(D) Some-Elements(d) Of-Type(d,D) Is-Relation(Desc:t,d) Is-Set(InColl) Is-Set(ExColl) Subset(InColl,Title) Subset(ExColl,Title) Disjoint(InColl,ExColl) No.of-Elements(InColl,≥,0) No.of-Elements(ExColl,≥,0) No.of-Elements(Title,=,0) </pre> <p>Definition:</p>

	<p>Is-Invariant(<i>C</i>) This expression represents the invariant for a class called <i>C</i>. The invariant has all the expressions that declare and predicate the conditions. These conditions must be always true about this class.</p>
<p>Example 7: An Event Specification</p>	<p>There is an invariant called <i>VM</i>. his invariant has the definition of <i>Member</i> set and the <i>Info</i> relation as shown below:</p> <pre> Is-Invariant(VM) Is-Type(M) Is-Set(Member) Of-Type(Member,M) No.of-Elements(Member,≥,0) All-Elements(m) Element-in-Set(m,Member) Is-Type(I) Some-Elements(i) Is-Relation(Info:m,i) </pre> <p>Based on this invariant there is an event (operation) called <i>VM!NewMember(m,i)</i>. The invariant name, <i>VM</i>, which appears in the event's name, is a reference to hidden information that previously defined. This event is used to register a new member, <i>m</i>, with information <i>i</i>. A specification of an event has 2 conditions; one condition (called pre-condition) is used to show things before changes such as initiated members of sets and the other condition (called post-condition) is used to show what has changed. The pre-condition (before applying the event) ensures that its input-argument, <i>i</i>, has type I, and that the other input-argument, <i>m</i>, is an identifier of type M which is not in <i>Member</i>. In the post-condition (after applying that event), <i>m</i> is associated with <i>i</i> by a relation called <i>Info</i> (Information) and it is in <i>Member</i>. Dashed names denote values that are changed- but only minimal changes are shown here; there is no need to say that other elements of <i>Member</i> and their associated <i>Info</i> values remain the same, because of the convention that "the rest stays unchanged".</p> <p>The following formal expression represents this statement.</p> <pre> Is-Event(VM!NewMember) Pre-Condition: Is-Type(I) Is-Element(i) Of-Type(i,I) Is-Type(M) Is-Set(Member) Of-Type(Member,M) No.of-Elements(Member,≥,0) Is-Element(m) Of-Type(m,M) Post-Condition: Element-in-Set(m,Member) Is-Relation(Info:m,i) </pre>

	<p>Definition:</p> <p>Is-Event(C!E) Pre-Condition:</p> <p>Post-Condition:</p> <p>This expression represents an event called <i>C!E</i> which specifies some allowable change-of-state for objects. Thus it always involves a POST-CONDITION and it may include an optional PRE-CONDITION as well. The ! is used to separate the event and the invariant names.</p>
--	---

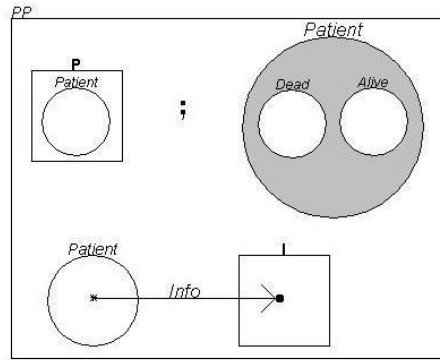
The editor that we developed allows the user to drag and drop from the toolbar and also allows them to perform some actions such as: to right-click on that diagram to see a popup menu with some actions, to left-click on that diagram to move it anywhere, and to press shift and left-click on that diagram to bring it to front. After each example three questions are presented to the participants. Each question is divided into two questions (part A and part B). Before answering, participants could return to the related example at any time. However, after answering part A they would get part B and after answering part B, the correct answer would be shown to the participant. Table B.3 represents the part A of the 21 questions for both groups.

Table B.3 The Part A - Questions for both the CD and the NL groups

	CD	NL
Training Q	<p>A circle, called B, is contained in another circle, called A. The area of A outside of B is shaded. The area of intersection between A and B is not. There is a dot in B and also a dot in A which is not in B. A single-open-rectangle, called C, contains these two circles.</p> <p>The produced diagram should look like this:</p> 	<p>A group, called B, is contained in another group, called A. The members of B are already members of A. However, if there is a member in A but not in B, then this is the only member there. B can have many members. An invariant, called C, contains these definitions.</p> <p>The produced formal expression should look like this:</p> <p>Is-Invariant(C) Is-Set(A) Is-Set(B) Subset(B,A) No.of-Elements(B,\geq,1) No.of-Elements(A,$=$,1)</p>

	<p>Please try to reproduce this diagram by using the on-screen editor.</p> <p>The question will be:</p> <p>From the given statement, construct a diagram that asserts this situation.</p>	<p>Please try to reproduce this formal expression by using the on-screen editor.</p> <p>The question will be:</p> <p>From the given statement, construct a formal expression that asserts this situation.</p>
Q1	<p>John has a particular property of being classified as a patient.</p> <p>From the given statement, construct a diagram that asserts this situation.</p>	<p>John has a particular property of being classified as a patient.</p> <p>From the given statement, construct a formal expression that asserts this situation.</p>
Q2	<p>There is a group of patients.</p> <p>From the given statement, construct a diagram that asserts this situation.</p>	<p>There is a group of patients.</p> <p>From the given statement, construct a formal expression that asserts this situation.</p>
Q3	<p>There is a group of patients and there is John who is not a member of that group. However, John and the group are sharing a particular property.</p> <p>From the given statement, construct a diagram that asserts this situation.</p>	<p>There is a group of patients and there is John who is not a member of that group. However, John and the group are sharing a particular property.</p> <p>From the given statement, construct a formal expression that asserts this situation.</p>
Q4	<p>Initially in the system, there are no patient records.</p> <p>From the given statement, construct a diagram that asserts this situation.</p>	<p>Initially in the system, there are no patient records.</p> <p>From the given statement, construct a formal expression that asserts this situation.</p>
Q5	<p>There is exactly one patient.</p> <p>From the given statement, construct a diagram that asserts this situation.</p>	<p>There is exactly one patient.</p> <p>From the given statement, construct a formal expression that asserts this situation.</p>
Q6	<p>There is at least one service.</p> <p>From the given statement, construct a diagram that asserts this situation.</p>	<p>There is at least one service.</p> <p>From the given statement, construct a formal expression that asserts this situation.</p>
Q7	<p>Some patients are alive (some are not).</p> <p>From the given statement, construct a diagram that asserts this situation.</p>	<p>Some patients are alive (some are not).</p> <p>From the given statement, construct a formal expression that asserts this situation.</p>
Q8	<p>Patients are either dead or alive (but not both).</p>	<p>Patients are either dead or alive (but not both).</p>

	From the given statement, construct a diagram that asserts this situation.	From the given statement, construct a formal expression that asserts this situation.
Q9	<p>Patients are dead or alive.</p> <p>From the given statement, construct a diagram that asserts this situation.</p>	<p>Patients are dead or alive.</p> <p>From the given statement, construct a formal expression that asserts this situation.</p>
Q10	<p>All health professionals have qualifications.</p> <p>From the given statement, construct a diagram that asserts this situation.</p>	<p>All health professionals have qualifications.</p> <p>From the given statement, construct a formal expression that asserts this situation.</p>
Q11	<p>All patients have some associated information.</p> <p>From the given statement, construct a diagram that asserts this situation.</p>	<p>All patients have some associated information.</p> <p>From the given statement, construct a formal expression that asserts this situation.</p>
Q12	<p>One patient called John is associated to some specific information.</p> <p>From the given statement, construct a diagram that asserts this situation.</p>	<p>One patient called John is associated to some specific information.</p> <p>From the given statement, construct a formal expression that asserts this situation.</p>
Q13	<p>A patient-record is either a note or a communication and it is identified by its author (a health-professional).</p> <p>From the given statement, construct a diagram that asserts this situation.</p>	<p>A patient-record is either a note or a communication and it is identified by its author (a health-professional).</p> <p>From the given statement, construct a formal expression that asserts this situation.</p>
Q14	<p>A patient called p is either alive or dead (but not both).</p> <p>From the given statement, construct a diagram that asserts this situation.</p>	<p>A patient called p is either alive or dead (but not both).</p> <p>From the given statement, construct a formal expression that asserts this situation.</p>
Q15	<p>Some services are subservices of other services.</p> <p>From the given statement, construct a diagram that asserts this situation.</p>	<p>Some services are subservices of other services.</p> <p>From the given statement, construct a formal expression that asserts this situation.</p>
Q16	<p>There is a concept of patient population which consists of a group of patients who have some associated information.</p>	<p>There is a concept of patient population which consists of a group of patients who have some associated information. Patients are either alive</p>

	<p>Patients are either alive or dead (but not both).</p> <p>From the given statement, construct a diagram that asserts this situation.</p>	<p>or dead (but not both).</p> <p>From the given statement, construct a formal expression that asserts this situation.</p>
Q17	<p>There is a concept of service system which consists of a group of services. Some services are subservices of other services.</p> <p>From the given statement, construct a diagram that asserts this situation.</p>	<p>There is a concept of service system which consists of a group of services. Some services are subservices of other services.</p> <p>From the given statement, construct a formal expression that asserts this situation.</p>
Q18	<p>There is a concept of health system which consists of a group of health professionals who have qualifications.</p> <p>From the given statement, construct a diagram that asserts this situation.</p>	<p>There is a concept of health system which consists of a group of health professionals who have qualifications.</p> <p>From the given statement, construct a formal expression that asserts this situation.</p>
Q19	<p>Given a concept called <i>PP</i>:</p>  <p>A new patient can be registered.</p> <p>From the given concept and statement, construct a diagram that asserts this situation.</p>	<p>Given a concept called <i>PP</i>:</p> <p>Is-Invariant(<i>PP</i>) Is-Type(<i>P</i>) Is-Set(<i>Patient</i>) Of-Type(<i>Patient</i>,<i>P</i>) Is-Set(<i>Alive</i>) Is-Set(<i>Dead</i>) Subset(<i>Alive</i>,<i>Patient</i>) Subset(<i>Dead</i>,<i>Patient</i>) Disjoint(<i>Alive</i>,<i>Dead</i>) No.of-Elements(<i>Patient</i>,=,0) No.of-Elements(<i>Alive</i>,≥,0) No.of-Elements(<i>Dead</i>,≥,0) All-Elements(<i>p</i>) Element-in-Set(<i>p</i>,<i>Patient</i>) Some-Elements(<i>i</i>) Of-Type(<i>i</i>,<i>I</i>) Is-Relation(<i>Info</i>:<i>p</i>,<i>i</i>)</p> <p>A new patient can be registered.</p> <p>From the given concept and statement, construct a formal expression that asserts this situation.</p>
Q20	<p>Given a concept called <i>PP</i>:</p>	<p>Given a concept called <i>PP</i>:</p>

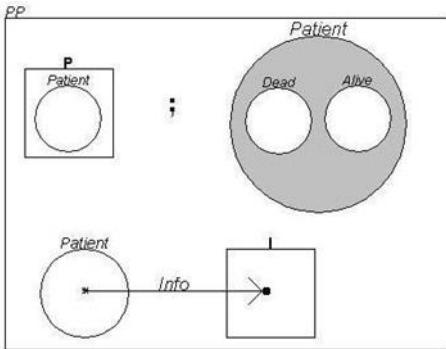
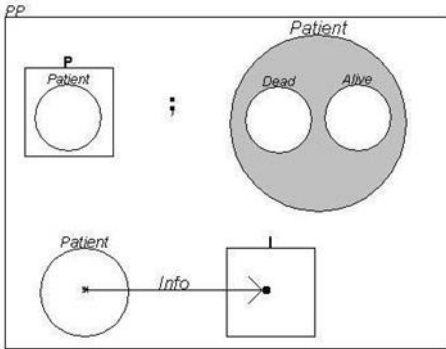
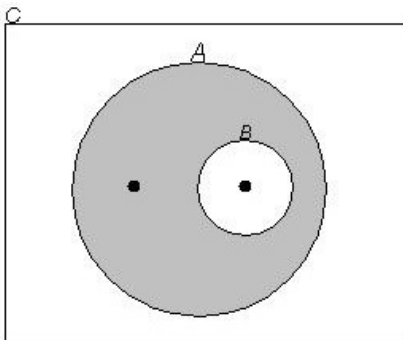
	 <p>Current information for a patient can be updated to some new value.</p> <p>From the given concept and statement, construct a diagram that asserts this situation.</p>	<p>Is-Invariant(PP) Is-Type(P) Is-Set($Patient$) Of-Type($Patient, P$) Is-Set($Alive$) Is-Set($Dead$) Subset($Alive, Patient$) Subset($Dead, Patient$) Disjoint($Alive, Dead$) No.of-Elements($Patient, =, 0$) No.of-Elements($Alive, \geq, 0$) No.of-Elements($Dead, \geq, 0$) All-Elements(p) Element-in-Set($p, Patient$) Some-Elements(i) Of-Type(i, I) Is-Relation($Info: p, i$)</p> <p>Current information for a patient can be updated to some new value.</p> <p>From the given concept and statement, construct a formal expression that asserts this situation.</p>
Q21	<p>Given a concept called PP:</p>  <p>A patient's death can be recorded.</p> <p>From the given concept and statement, construct a diagram that asserts this situation.</p>	<p>Given a concept called PP:</p> <p>Is-Invariant(PP) Is-Type(P) Is-Set($Patient$) Of-Type($Patient, P$) Is-Set($Alive$) Is-Set($Dead$) Subset($Alive, Patient$) Subset($Dead, Patient$) Disjoint($Alive, Dead$) No.of-Elements($Patient, =, 0$) No.of-Elements($Alive, \geq, 0$) No.of-Elements($Dead, \geq, 0$) All-Elements(p) Element-in-Set($p, Patient$) Some-Elements(i) Of-Type(i, I) Is-Relation($Info: p, i$)</p> <p>A patient's death can be recorded.</p> <p>From the given concept and statement, construct a formal expression that asserts this situation.</p>

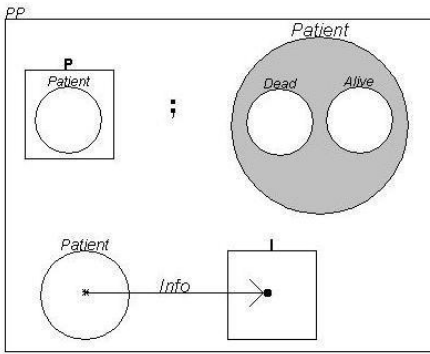
Table B.4 represents the part B of the 21 questions for both groups.

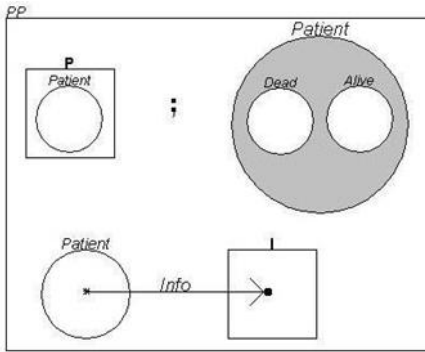
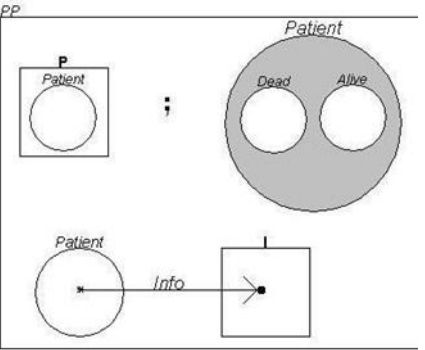
Table B.4 The Part B - Questions for both the CD and the NL groups

	CD	NL
Training Q	<p>There is an element in a set called A and there is another element in a set called B. The intersection of A and B is B. If the element is in A but not in B, then it is the only element there. All of this is defined in a class called C.</p> <p>The produced diagram should look like this:</p>  <p>Please try to reproduce this diagram by using the on-screen editor.</p> <p>The question will be:</p> <p>From the given statement, construct a diagram that asserts this situation.</p>	<p>There is an element in a set called A and there is another element in a set called B. The intersection of A and B is B. If the element is in A but not in B, then it is the only element there. All of this is defined in a class called C.</p> <p>The produced formal expression should look like this:</p> <pre>Is-Invariant(C) Is-Set(A) Is-Set(B) Subset(B,A) No.of-Elements(B,≥,1) No.of-Elements(A,=,1)</pre> <p>Please try to reproduce this formal expression by using the on-screen editor</p> <p>The question will be:</p> <p>From the given statement, construct a formal expression that asserts this situation.</p>
Q1	<p>There is an element labelled <i>John</i> which is of type P.</p> <p>From the given statement, construct a diagram that asserts this situation.</p>	<p>There is an element called <i>John</i> which is of type P.</p> <p>From the given statement, construct a formal expression that asserts this situation.</p>
Q2	<p>There is a set of patients labelled <i>Patient</i>.</p> <p>From the given statement, construct a diagram that asserts this situation.</p>	<p>There is a set of patients called <i>Patient</i>.</p> <p>From the given statement, construct a formal expression that asserts this situation.</p>
Q3	<p>There is a set of patients labelled <i>Patient</i> and there is an element labeled <i>John</i> which is not in the set <i>Patient</i>. <i>John</i> and elements in <i>Patient</i> are of type P.</p> <p>From the given statement, construct a diagram that asserts this situation.</p>	<p>There is a set of patients called <i>Patient</i> and there is an element called <i>John</i> which is not in the set <i>Patient</i>. <i>John</i> and elements in <i>Patient</i> are of type P.</p> <p>From the given statement, construct a formal expression that asserts this situation.</p>

Q4	<p><i>Patient-Record</i> is an empty set.</p> <p>From the given statement, construct a diagram that asserts this situation.</p>	<p><i>Patient-Record</i> is an empty set.</p> <p>From the given statement, construct a formal expression that asserts this situation.</p>
Q5	<p>A set called <i>Patient</i> has exactly one element.</p> <p>From the given statement, construct a diagram that asserts this situation.</p>	<p>A set called <i>Patient</i> has exactly one element.</p> <p>From the given statement, construct a formal expression that asserts this situation.</p>
Q6	<p>A set called <i>Service</i> has at least one element.</p> <p>From the given statement, construct a diagram that asserts this situation.</p>	<p>A set called <i>Service</i> has at least one element.</p> <p>From the given statement, construct a formal expression that asserts this situation.</p>
Q7	<p>A set called <i>Alive</i> is a subset of a set called <i>Patient</i>.</p> <p>From the given statement, construct a diagram that asserts this situation.</p>	<p>A set called <i>Alive</i> is a subset of a set called <i>Patient</i>.</p> <p>From the given statement, construct a formal expression that asserts this situation.</p>
Q8	<p>A set called <i>Alive</i> and a set called <i>Dead</i> are disjoint and their union is a set called <i>Patient</i>.</p> <p>From the given statement, construct a diagram that asserts this situation.</p>	<p>A set called <i>Alive</i> and a set called <i>Dead</i> are disjoint and their union is a set called <i>Patient</i>.</p> <p>From the given statement, construct a formal expression that asserts this situation.</p>
Q9	<p>A set called <i>Alive</i> and a set called <i>Dead</i> are both subsets of a set called <i>Patient</i>.</p> <p>From the given statement, construct a diagram that asserts this situation.</p>	<p>A set called <i>Alive</i> and a set called <i>Dead</i> are both subsets of a set called <i>Patient</i>.</p> <p>From the given statement, construct a formal expression that asserts this situation.</p>
Q10	<p>Each member of a set called <i>HProf</i> is associated with some set whose elements are of type Q by relation called <i>Qualif</i>.</p> <p>From the given statement, construct a diagram that asserts this situation.</p>	<p>Each member of a set called <i>HProf</i> is associated with some set whose elements are of type Q by relation called <i>Qualif</i>.</p> <p>From the given statement, construct a formal expression that asserts this situation.</p>
Q11	<p>Each element of a set called <i>Patient</i> is associated with an element of type I by a relation called <i>Info</i>.</p>	<p>Each element of a set called <i>Patient</i> is associated with an element of type I by a relation called <i>Info</i>.</p>

	From the given statement, construct a diagram that asserts this situation.	From the given statement, construct a formal expression that asserts this situation.
Q12	<p><i>John</i> who is an element of a set called <i>Patient</i> is not associated with an element called <i>i</i> of type I by a relation called <i>Info</i>. However, he is associated with some other element of type I.</p> <p>From the given statement, construct a diagram that asserts this situation.</p>	<p><i>John</i> who is an element of a set called <i>Patient</i> is not associated with an element called <i>i</i> of type I by a relation called <i>Info</i>. However, he is associated with some other element of type I.</p> <p>From the given statement, construct a formal expression that asserts this situation.</p>
Q13	<p>There is a set called <i>Note</i> and another set called <i>Communication</i> (<i>Comm</i>). They are the only subsets of a set called <i>Patient-Record</i>. Each record, which could be in <i>Note</i> or in <i>Comm</i>, is associated with some element in a set called <i>HProf</i> (health-professional) by a relation called <i>Author</i>.</p> <p>From the given statement, construct a diagram that asserts this situation.</p>	<p>There is a set called <i>Note</i> and another set called <i>Communication</i> (<i>Comm</i>). They are the only subsets of a set called <i>Patient-Record</i>. Each record, which could be in <i>Note</i> or in <i>Comm</i>, is associated with some element in a set called <i>HProf</i> (health-professional) by a relation called <i>Author</i>.</p> <p>From the given statement, construct a formal expression that asserts this situation.</p>
Q14	<p>There is <i>p</i> which is an element of a set called <i>Patient</i>. <i>p</i> is either in one of two subsets; <i>Alive</i> or <i>Dead</i> (but not both).</p> <p>From the given statement, construct a diagram that asserts this situation.</p>	<p>There is <i>p</i> which is an element of a set called <i>Patient</i>. <i>p</i> is either in one of two subsets; <i>Alive</i> or <i>Dead</i> (but not both).</p> <p>From the given statement, construct a formal expression that asserts this situation.</p>
Q15	<p>There is a set called <i>Service</i> which contains another set called <i>Sub</i> (subservice). Each element of <i>Sub</i> is related to some other service which could be another subservice by a relation called <i>RelatedTo</i>.</p> <p>From the given statement, construct a diagram that asserts this situation.</p>	<p>There is a set called <i>Service</i> which contains another set called <i>Sub</i> (subservice). Each element of <i>Sub</i> is related to some other service which could be another subservice by a relation called <i>RelatedTo</i>.</p> <p>From the given statement, construct a formal expression that asserts this situation.</p>
Q16	The invariant of a <i>Patient Population</i> class which called <i>PP</i> maintains a set called <i>Patient</i> whose elements are of type P . Each patient has associated	The invariant of a <i>Patient Population</i> class which called <i>PP</i> maintains a set called <i>Patient</i> whose elements are of type P . Each patient has associated <i>Information</i> of type I and is either

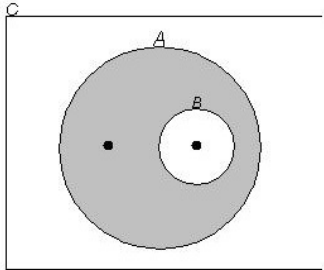
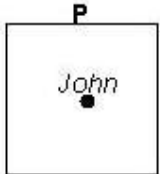
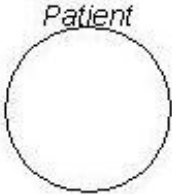
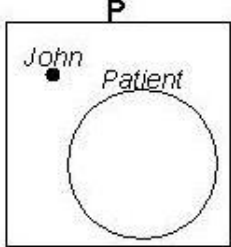
	<p>Information of type I and is either <i>Alive</i> or in the end <i>Dead</i> but not both.</p> <p>From the given statement, construct a diagram that asserts this situation.</p>	<p><i>Alive</i> or in the end <i>Dead</i> but not both.</p> <p>From the given statement, construct a formal expression that asserts this situation.</p>
Q17	<p>The invariant of a <i>Service System</i> class which called <i>SS</i> maintains a set called <i>Service</i> whose elements of type S. <i>Service</i> set contains another set called <i>Sub</i> (subservice). Each element of <i>Sub</i> is <i>related to</i> some other service which could be another subservice.</p> <p>From the given statement, construct a diagram that asserts this situation.</p>	<p>The invariant of a <i>Service System</i> class which called <i>SS</i> maintains a set called <i>Service</i> whose elements of type S. <i>Service</i> set contains another set called <i>Sub</i> (subservice). Each element of <i>Sub</i> is <i>related to</i> some other service which could be another subservice.</p> <p>From the given statement, construct a formal expression that asserts this situation.</p>
Q18	<p>The invariant of a <i>Health System</i> class called <i>HS</i> maintains a set called <i>HProf</i> whose elements are of type H. Each <i>HProf</i> has associated set of type Q by a relation called <i>Qualif</i>.</p> <p>From the given statement, construct a diagram that asserts this situation.</p>	<p>The invariant of a <i>Health System</i> class called <i>HS</i> maintains a set called <i>HProf</i> whose elements are of type H. Each <i>HProf</i> has associated set of type Q by a relation called <i>Qualif</i>.</p> <p>From the given statement, construct a formal expression that asserts this situation.</p>
Q19	<p>Given an invariant called <i>PP</i>:</p>  <p>An event called '<i>RegisterPatient(p,i)</i>' of <i>PP</i> class is performed to register a new patient called <i>p</i> with associated information if and only if <i>p</i>, which is not a member in the <i>Patient</i> set yet, is of type P and <i>i</i> is of type I. By registering a new patient, he/she will be in <i>Alive</i> set along with his/her <i>information</i>.</p> <p>From the given invariant and statement,</p>	<p>Given an invariant called <i>PP</i>:</p> <pre> Is-Invariant(PP) Is-Type(P) Is-Set(Patient) Of-Type(Patient,P) Is-Set(Alive) Is-Set(Dead) Subset(Alive,Patient) Subset(Dead,Patient) Disjoint(Alive,Dead) No.of-Elements(Patient,=,0) No.of-Elements(Alive,≥,0) No.of-Elements(Dead,≥,0) All-Elements(p) Element-in-Set(p,Patient) Some-Elements(i) Of-Type(i,I) Is-Relation(Info:p,i) </pre> <p>An event called '<i>RegisterPatient(p,i)</i>' of <i>PP</i> class is performed to register a new patient called <i>p</i> with associated information if and only if <i>p</i>, which is not a member in the <i>Patient</i> set yet, is of type P and <i>i</i> is of type I. By registering a new patient, he/she will be in <i>Alive</i> set along</p>

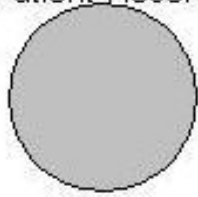
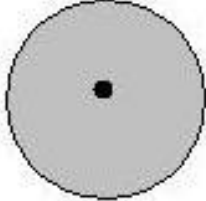
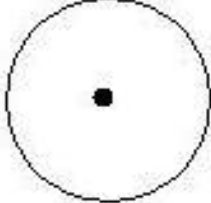
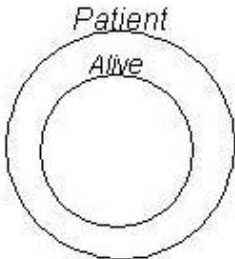
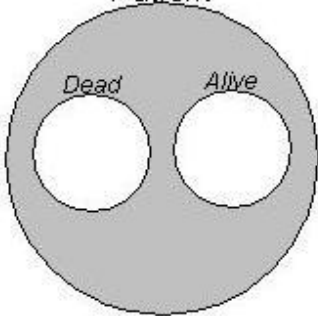
	construct a diagram that asserts this situation.	with his/her <i>information</i> . From the given invariant and statement, construct a formal expression that asserts this situation.
Q20	<p>Given an invariant called <i>PP</i>:</p>  <p>An event called '<i>UpdatePatientInformation(p,i)</i>' of class called <i>PP</i> can be used to update the current Information for a patient <i>p</i> to some new value <i>I</i> if and only if the current value of Information differs from the new value. Both values are of type <i>I</i>.</p> <p>From the given invariant and statement, construct a diagram that asserts this situation.</p>	<p>Given an invariant called <i>PP</i>:</p> <pre> Is-Invariant(PP) Is-Type(P) Is-Set(Patient) Of-Type(Patient,P) Is-Set(Alive) Is-Set(Dead) Subset(Alive,Patient) Subset(Dead,Patient) Disjoint(Alive,Dead) No.of-Elements(Patient,=,0) No.of-Elements(Alive,≥,0) No.of-Elements(Dead,≥,0) All-Elements(p) Element-in-Set(p,Patient) Some-Elements(i) Of-Type(i,I) Is-Relation(Info:p,i) </pre> <p>An event called '<i>UpdatePatientInformation(p,i)</i>' of class called <i>PP</i> can be used to update the current Information for a patient <i>p</i> to some new value <i>I</i> if and only if the current value of Information differs from the new value. Both values are of type <i>I</i>.</p> <p>From the given invariant and statement, construct a formal expression that asserts this situation.</p>
Q21	<p>Given an invariant called <i>PP</i>:</p>  <p>An event called '<i>RecordDeath(p)</i>' of class called <i>PP</i> can be used to record the death of a patient called <i>p</i> by moving the</p>	<p>Given an invariant called <i>PP</i>:</p> <pre> Is-Invariant(PP) Is-Type(P) Is-Set(Patient) Of-Type(Patient,P) Is-Set(Alive) Is-Set(Dead) Subset(Alive,Patient) Subset(Dead,Patient) Disjoint(Alive,Dead) No.of-Elements(Patient,=,0) No.of-Elements(Alive,≥,0) No.of-Elements(Dead,≥,0) All-Elements(p) Element-in-Set(p,Patient) Some-Elements(i) Of-Type(i,I) Is-Relation(Info:p,i) </pre>

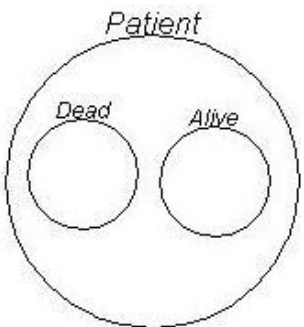

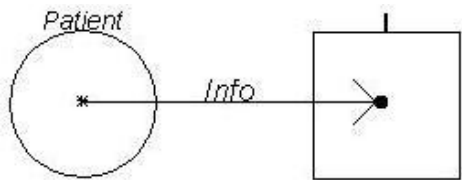
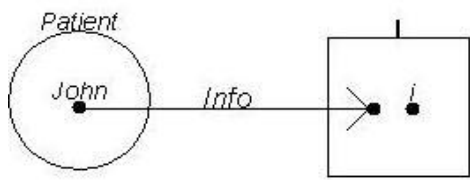
	<p>patient membership from <i>Alive</i> to <i>Dead</i> set.</p> <p>From the given invariant and statement, construct a diagram that asserts this situation.</p>	<p>An event called '<i>RecordDeath(p)</i>' of class called <i>PP</i> can be used to record the death of a patient called <i>p</i> by moving the patient membership from <i>Alive</i> to <i>Dead</i> set.</p> <p>From the given invariant and statement, construct a formal expression that asserts this situation.</p>
--	---	--

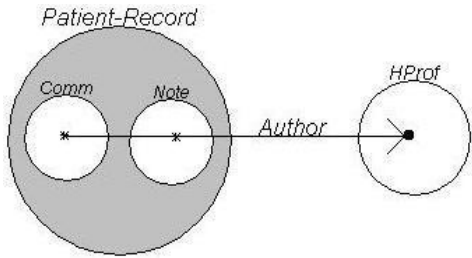
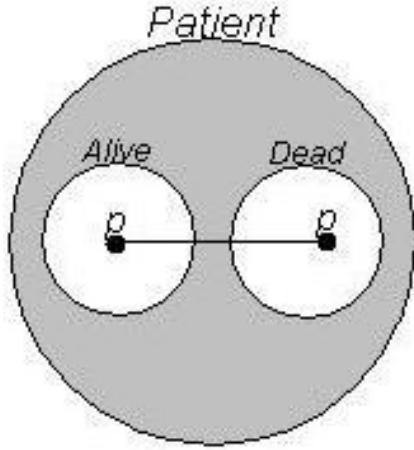
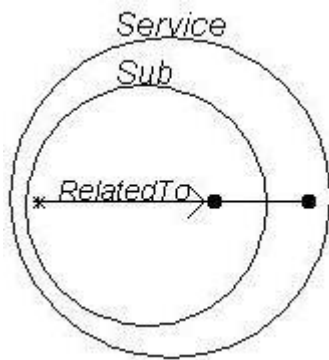
Table B.5 represents the correct answers for both CD and NL groups.

Table B.5 The correct answer

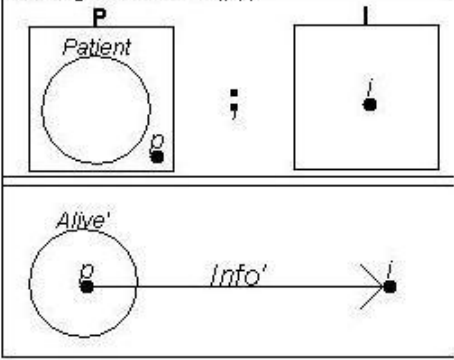
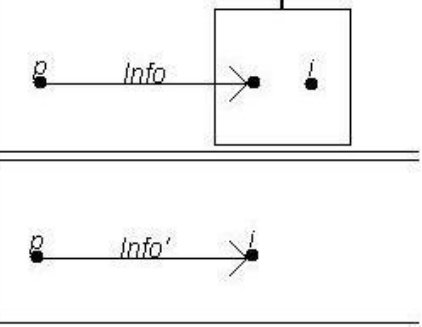
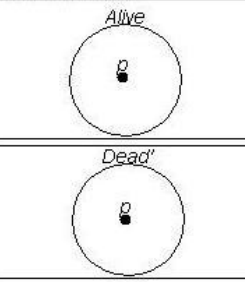
	CD	NL
Training Q		<p>Is-Invariant(C)</p> <p>Is-Set(A)</p> <p>Is-Set(B)</p> <p>Subset(B,A)</p> <p>No.of-Elements(B, \geq, 1)</p> <p>No.of-Elements(A, =, 1)</p>
Q1		<p>Is-Type(P)</p> <p>Is-Element(John)</p> <p>Of-Type(John,P)</p>
Q2		<p>Is-Set(Patient)</p>
Q3		<p>Is-Type(P)</p> <p>Is-Set(Patient)</p> <p>Is-Element(John)</p> <p>Of-Type(Patient,P)</p> <p>Of-Type(John,P)</p>

Q4	<i>Patient-Record</i> 	Is-Set(Patient) No.of-Elements(Patient, =, 0)
Q5	<i>Patient</i> 	Is-Set(Patient) No.of-Elements(Patient, =, 1)
Q6	<i>Service</i> 	Is-Set(Service) No.of-Elements(Service, ≥, 1)
Q7	<i>Patient</i> <i>Alive</i> 	Is-Set(Patient) Is-Set(Alive) Subset(Alive, Patient) No.of-Elements(Patient, ≥, 0) No.of-Elements(Alive, ≥, 0)
Q8	<i>Patient</i> <i>Dead</i> <i>Alive</i> 	Is-Set(Patient) Is-Set(Alive) Is-Set(Dead) Subset(Alive, Patient) Subset(Dead, Patient) Disjoint(Alive, Dead) No.of-Elements(Patient, =, 0) No.of-Elements(Alive, ≥, 0) No.of-Elements(Dead, ≥, 0)

Q9		<p> Is-Set(Patient) Is-Set(Alive) Is-Set(Dead) Subset(Alive,Patient) Subset(Dead,Patient) Disjoint(Alive,Dead) No.of-Elements(Patient,\geq,0) No.of-Elements(Alive,\geq,0) No.of-Elements(Dead,\geq,0) </p>
Q10		<p> Is-Set(HProf) All-Elements(h) Element-in-Set(h,HProf) Is-Type(Q) Is-Set(q) Of-Type(q,Q) Is-Relation(Qualif:h,q) </p>
Q11		<p> Is-Set(Patient) All-Elements(p) Element-in-Set(p,Patient) Is-Type(I) Some-Elements(i) Of-Type(i,I) Is-Relation(Info:p,i) </p>
Q12		<p> Is-Set(Patient) Is-Element(John) Element-in-Set(John,Patient) Is-Type(I) Is-Element(i) Of-Type(i,I) Some-Elements(s) Of-Type(s,I) Is-Relation(Info:John,s) </p>

Q13		<p> Is-Set(Patient-Record) Is-Set(HProf) Is-Set(Note) Is-Set(Comm) Subset(Note,Patient-Record) Subset(Comm,Patient-Record) Disjoint(Note,Comm) All-Elements(n) Element-in-Set(n,Note) All-Elements(c) Element-in-Set(c,Comm) Same-Element(n,c) No.of-Elements(Patient-Record,=,0) Some-Elements(h) Element-in-Set(h,HProf) Is-Relation(Author:n,h) </p>
Q14		<p> Is-Set(Patient) Is-Set(Alive) Is-Set(Dead) Subset(Alive,Patient) Subset(Dead,Patient) Disjoint(Alive,Dead) No.of-Elements(Patient,=,0) No.of-Elements(Alive,≥,1) No.of-Elements(Dead,≥,1) Is-Element(p) Element-in-Set(p,Alive) Element-in-Set(p,Dead) Same-Element(Alive.p,Dead.p) </p>
Q15		<p> Is-Set(Service) Is-Set(Sub) Subset(Sub,Service) All-Elements(s) Element-in-Set(s,Sub) Some-Elements(r) Element-in-Set(r,Sub) Some-Elements(t) Element-in-Set(t,Service) Is-Relation(RelatedTo:s,r) Same-Element(r,t) </p>

Q16		<p> Is-Invariant(PP) Is-Type(P) Is-Set(Patient) Of-Type(Patient,P) Is-Set(Alive) Is-Set(Dead) Subset(Alive,Patient) Subset(Dead,Patient) Disjoint(Alive,Dead) No.of-Elements(Patient,=,0) No.of-Elements(Alive,≥,0) No.of-Elements(Dead,≥,0) All-Elements(p) Element-in-Set(p,Patient) Some-Elements(i) Of-Type(i,I) Is-Relation(Info:p,i) </p>
Q17		<p> Is-Invariant(SS) Is-Type(S) Is-Set(Service) Of-Type(Service,S) Is-Set(Sub) Subset(Sub,Service) All-Elements(s) Element-in-Set(s,Sub) Some-Elements(r) Element-in-Set(r,Sub) Some-Elements(t) Element-in-Set(t,Service) Is-Relation(RelatedTo:s,r) Same-Element(r,t) </p>
Q18		<p> Is-Invariant(HS) Is-Type(H) Is-Set(HProf) Of-Type(HProf,H) Is-Type(Q) Is-Set(q) Of-Type(q,Q) All-Elements(h) Element-in-Set(h,HProf) Is-Relation(Qualif:h,q) </p>

Q19	<p><i>PP!RegisterPatient(p,i)</i></p> 	<p>Is-Event(<i>PP!RegisterPatient</i>) Pre-Condition: Is-Type(<i>I</i>) Is-Element(<i>i</i>) Of-Type(<i>i,I</i>) Is-Type(<i>P</i>) Is-Set(<i>Patient</i>) Of-Type(<i>Patient,P</i>) Is-Element(<i>p</i>) Of-Type(<i>p,P</i>) Post-Condition: Element-in-Set(<i>p,Alive</i>) Is-Relation(<i>Info:p,i</i>)</p>
Q20	<p><i>PP!UpdatePatientInfo(p,i)</i></p> 	<p>Is-Event(<i>PP!UpdatePatientInfo</i>) Pre-Condition: Is-Type(<i>I</i>) Is-Element(<i>i</i>) Of-Type(<i>i,I</i>) Some-Elements(<i>s</i>) Of-Type(<i>s,I</i>) Is-Relation(<i>Info:p,s</i>) Post-Condition: Is-Relation(<i>Info:p,i</i>)</p>
Q21	<p><i>PP!RecordDeath</i></p> 	<p>Is-Event(<i>PP!RecordDeath</i>) Pre-Condition: Element-in-Set(<i>p,Alive</i>) Post-Condition: Element-in-Set(<i>p,Dead</i>)</p>